Intel® 82801DB I/O Controller Hub (ICH4): AC '97

Programmers Reference Manual (PRM)

May 2002

Revision 001

Document Number: 251373-001

Contents

Figures

Tables

Revision History

Revision	Description	Date
-001	• Initial Release	May 2002

Reference Documents and Information Sources

Document Name or Information Source	Available From
Intel® ICH4 AC '97 External Design Specification Revision 1.0	Intel
Audio Codec '97 Specification, Revision 2.1 , Revision 2.2 and Revision 2.3	Intel
Communications and Networking Riser Specification, Version 1.0 and 1.2	Intel
Microsoft* Windows* Driver Development Kits – http://www.microsoft.com/ddk	Microsoft
Microsoft* Windows* Driver and Hardware Development -- http://www.microsoft.com/hwdev	Microsoft

Applicable Components

Device Name	Vendor ID	Device ID	Subsystem Vendor ID	Subsystem Device ID	Base Class Code	Sub-Class Code	Prog. Interface	Revision ID	Bus Number (PCI Addr)	Device Number	Function Number	Microsoft* PNP Device Node ID	Intel Desired Device Description (INF name) Name for: Windows* 95 Windows* 98
Intel® ICH	8086	2415	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	04h	01h	00h	ALL	00h	1Fh	5	PCI\VEN_8086&DEV_2415 (subsystem will also provide additional information)	Intel® 82801AA AC '97 Audio Controller (displayed by driver provider's INF)
Intel ICH	8086	2416	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	07h	03h	00h	ALL	00h	1Fh	6	PCI\VEN_8086&DEV_2416 (subsystem will also provide additional information)	Intel 82801AA AC '97 Modem Controller (displayed by driver provider's INF)
Intel ICH-0	8086	2425	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	04h	01h	00h	ALL	00h	1Fh	5	PCI\VEN_8086&DEV_2425 (subsystem will also provide additional information)	Intel® 82801AB AC '97 Audio Controller (displayed by driver provider's INF)
Intel ICH-0	8086	2426	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	07h	03h	00h	ALL	00h	1Fh	6	PCI\VEN_8086&DEV_2426 (subsystem will also provide additional information)	Intel 82801AB AC '97 Modem Controller (displayed by driver provider's INF)
Intel "ICH3	8086	2445	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	04h	01h	00h	ALL	00h	1Fh	5	PCI\VEN_8086&DEV_2445 (subsystem will also provide additional information)	Intel® "ICH3" AC '97 Audio Controller (displayed by driver provider's INF)
Intel "ICH3	8086	2446	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	04h	01h	00h	ALL	00h	1Fh	6	PCI\VEN_8086&DEV_2446 (subsystem will also provide additional information)	Intel® ICH3 DT/Server /Mobile/Low End" AC '97 Modem Controller (displayed by driver provider's INF)
Intel "ICH4	8086	24C5	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	04h	01h	00h	ALL	00h	1Fh	5	PCI\VEN_8086&DEV_24C5 (subsystem will also provide additional information)	Intel® "ICH4" AC '97 Audio Controller (displayed by driver provider's INF)

Device Name	Vendor ID	Device ID	Subsystem Vendor ID	Subsystem Device ID	Base Class Code	Sub-Class Code	Prog. Interface	Revision ID	Bus Number (PCI Addr)	Device Number	Function Number	Microsoft* PNP Device Node ID	Intel Desired Device Description (INF name) Name for: Windows* 95 Windows* 98
Intel "ICH4	8086	24C6	Default is 00h. Value of this register varies according to the system	Default is 00h. Value of this register varies according to the system	04h	01h	00h	ALL	00h	1Fh	6	PCI\VEN_8086&DEV_24C6 (subsystem will also provide additional information)	Intel® ICH4 DT/Server /Mobile/Low End" AC '97 Modem Controller (displayed by driver provider's INF)

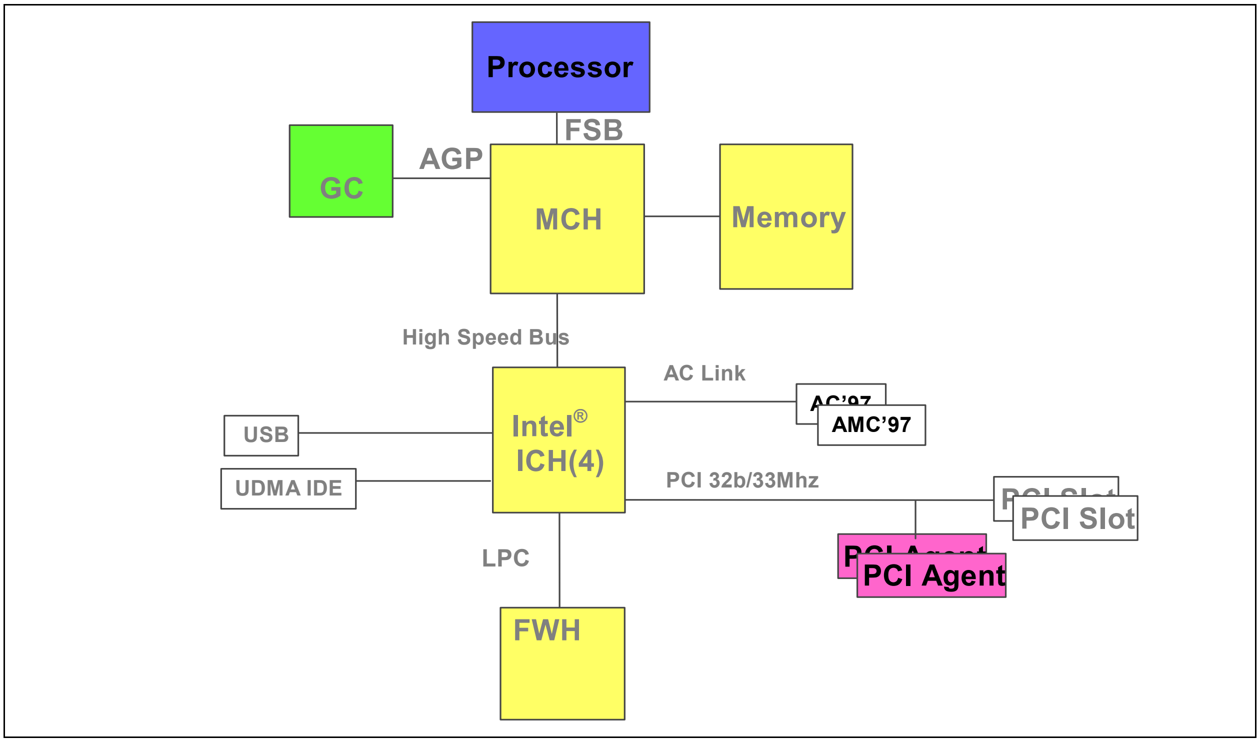
This page is intentionally left blank.

1. About This Document

This document was prepared to assist Independent Hardware Vendors (IHV) and Independent Software Vendors in supporting the Intel® 82801DB I/O Controller Hub (ICH4) AC '97 Digital Controller feature set. This document also applies to the previous generation of Intel I/O Controller Hub components. New features for the ICH4 which are not supported in the ICH component have been distinguished by gray shading to improve readability. Please refer to the *Applicable Component Table*, above. This document also describes the general requirements to develop an audio mini-port driver that will make use of the AC '97 audio interface. The primary purpose of this document is to supplement the information provided in the *Intel® I/O Controller Hub 4 (Intel® ICH4) AC'97 External Design Specification (EDS)* for use by IHVs and Intel customers developing their own driver interface.

This document also describes functions that the BIOS or Operating Systems (OS) must perform in order to ensure correct and reliable operation of the platform. This document will be supplemented from time to time with specification updates. The specification updates contain information relating to the latest programming changes. Check with your Intel representative for availability of specification updates.

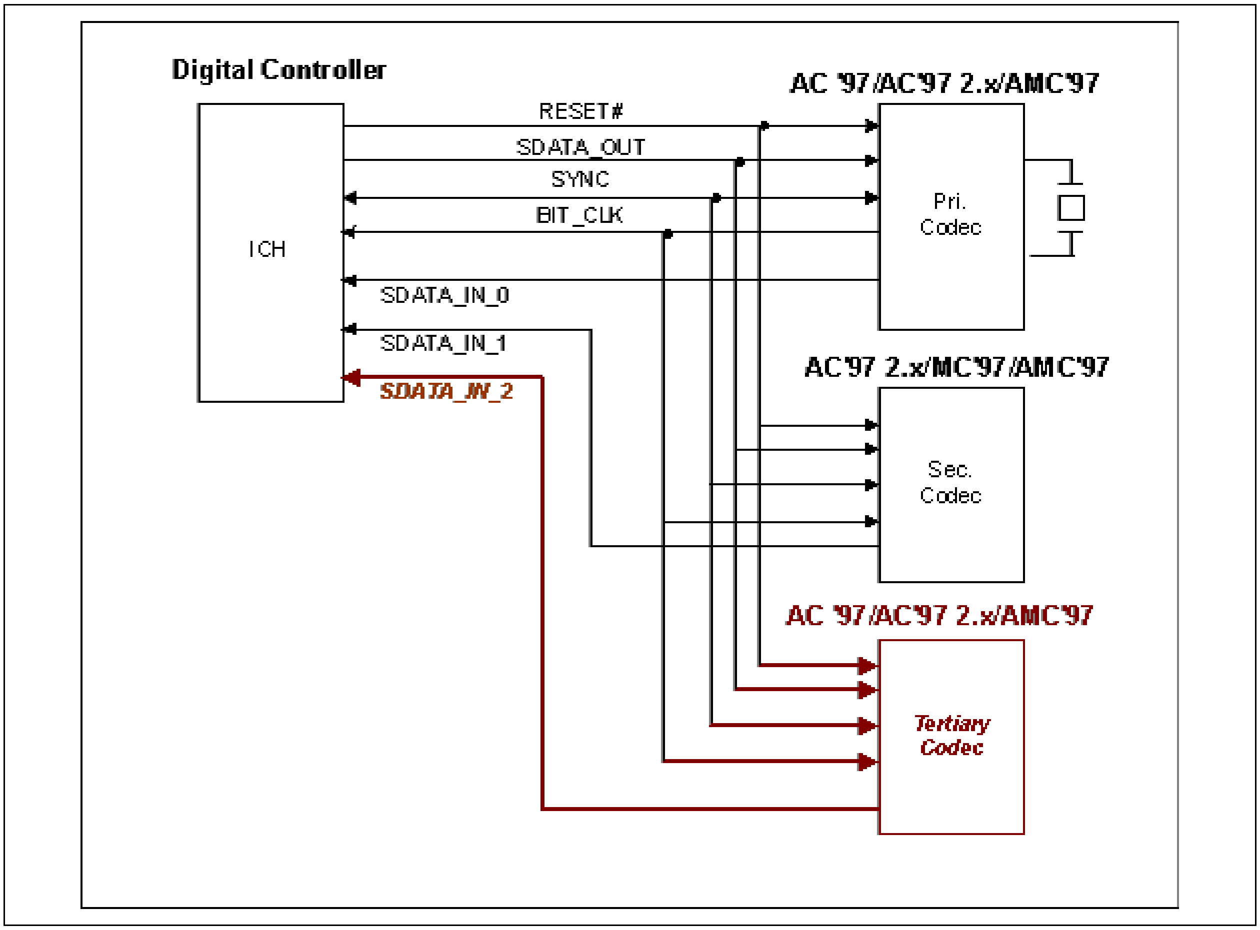
Figure 1-1. Block Diagram of Platform Chipset with Intel® ICH4 Component



In this document, "ICH4" stands for I/O Controller Hub4. The ICH4 provides an AC '97-compliant controller. References to the "AC '97 Component Specification" refer to the *Audio Codec '97 Specification*, Revision 2.1 and *Audio Codec '97 Specification*, Revision 2.2. The ICH4 AC'97 Digital Controller implementation interfaces to AC '97 Component Specification, Revision 2.1-compliant codecs. The ICH4 supports up to three AC '97 Component Specification compliant codec's on the AC-link interface. The figure below represents the typical configuration for the ICH4 AC '97 Controller and companion codecs.

1.1. Overview

Figure 1-3. Intel® ICH4 AC '97 Controller Connection to Its Companion Codecs



This document is limited to specify the software requirements and driver interface for the ICH4 AC '97 Digital Controller. Where possible, this document has pointers to additional considerations for supporting future proliferation or derivatives of the ICH4 Digital Controller. However, considerations for these future devices are subject to change. This document is base on the Intel® I/O Controller Hub (ICH4) AC'97 Software external Arquitecture Specifications (SAS) Rev 0.8

1.2. Intel® ICH4 AC '97 Controller Compatibility

The ICH4 AC '97 Controller is fully compatibility with the features found in the ICH1/2/3 versions. This allows for current drivers developed by ISVs and IHVs to work without modifications. See Section 1.2.4 . The ICH4 however, provides new capabilities not found in some of earlier ICH family of components. The following matrix provides a description of the available features for each of the ICHx components generation. This document specifically addresses new features on ICH4 while maintaining the original programming model reference for new developers working directly with ICH4 and not previously exposed to the ICH component.

Table 1-1. Audio Features Distribution Matrix

AC '97 Audio Controller Features	ICH	ICH2	ICH3	ICH4
16 bits Stereo PCM Output	☒	☒	☒	☒
16 bits Stereo PCM Input	☒	☒	☒	☒
16 bits Microphone Input	☒	☒	☒	☒
GPIO and Interrupt Support	☒	☒	☒	☒
Two 2.1/2.2 Codec Support	☒	☒	☒	☒
16 bits 2/4/6 Ch. Surround PCM Output		☒	☒	☒
20 bits 2/4/6 Ch. Surround PCM Output				☒
Dedicated S/P DIF DMA Output Ch.				☒
Third 2.1/2.2 Codec Support				☒
Memory Map Control and Status				☒
Second 16 bits Stereo PCM Input				☒
Second 16 bits Microphone Input				☒
PCI 2.2 Power Management				☒

The ICH4 AC '97 Audio Controller provides a set of new features that require significant software support. The following paragraphs provide a summary of these features.

The modem support infrastructure has not been changed in any generation of the I/O Controller Hub starting with the ICH.

1.2.1. Third AC '97 Component Specification 2.1 and AC '97 Component Specification 2.2 Compliant Codecs

The AC '97 Component Specification provides capability for up to four SDATA_IN signals for equal number of codec support. The ICH4 AC '97 Controller provides support for up to three codecs to allow for Audio channel expansion without sacrificing the Modem Codec (MC) support. Also, the third codec capability enables a better mobile docking infrastructure.

1.2.2. Dedicated S/P DIF DMA Output Channel

The AC '97 Specification Revision 2.2 provides the capability of steering an S/P DIF stream into the PCM channels for pass-through to a CE audio amplifier. The ICH4 provides this capability with an independent DMA channel that allows for a stream autonomous from the PCM audio available in other channels. This opens the possibility for an AC3 stream provided by DVD playback independent from system audio messages.

1.2.3. 20 Bits Surround PCM Output

The AC '97 Component Specification provides a maximum bit resolution of 20 bits per sample. The ICH4 AC '97 Controller DMA Engine fully exploits this capability to improve the audio output quality.

1.2.4. Memory Map Status and Control Registers

The ICH4 introduces a new PCI Memory Base Address Register that allows for higher performance access to the controller registers while expanding the register space to access the new third codec support mechanism. All access and features can know be access via this new Memory BAR making the I/O Bar capabilities obsolete. However, the ICH4 controller maintains the I/O BAR capability to allow for the reuse of legacy code maintaining backward compatibility to deployed driver binaries.

Note: This document describes the programming interface using the new Memory BAR registers unless otherwise indicated.

The default configuration for ICH4 Audio function is to use the PCI Memory Base Address Register. The I/O BAR is therefore disabled unless system BIOS enables the simultaneous backward compatible capability on register:

Device 31 Function 5 Audio			
Offset	**Register**	**Default**	**Comments**
41h	CFG Configuration	00h	When cleared, the I/O space BARs at offset 10h and 14h become read only registers. This is the default state for the I/O BARs. Initialized by BIOS when backward I/O Bar compatibility is required Memory BARs are always enable.

1.2.5. Second Independent Input DMA Engines

The ICH4 provides two new sets of input DMA engines that allow for the secondary or tertiary codecs to provide recording PCM data streams on the primary codec while simultaneously providing recording capabilities from the secondary or tertiary codec. A typical application is to provide independent input stream in am mobile docking configuration where an audio codec is located in the base system (notebook unit) and the secondary or tertiary codec is located in a docking unit for desktop replacement. The new DMA engines provide the infrastructure for s/w to select the input stream from either source for stereo or microphone recording. Also the capability of simultaneous input streams opens the possibilities for more futuristic applications where a microphone array can be created using two codecs.

1.2.6. PCI Local Bus Specification, Revision 2.2 Power Management

The ICH4 introduces *PCI Local Bus Specification*, Revision 2.2-compliant power management registers that allows for better OS power management support with reduced overhead to the BIOS programmers using ACPI control methodologies.

1.3. General Requirements

It is assumed that the reader has a working knowledge of AC '97 architecture and the ICH4 AC'97 Controller implementation. Also, the reader should have an understanding of audio driver development for the target operating systems.

This document outlines the software specification for the AC '97 Digital Controller, and also includes details on the development of an audio device driver that will be used in Windows* 98 family, Windows NT*, Windows* 2000 and Windows* XP, based on the Windows* Driver Model (WDM) interface.

2. Intel® ICH4 AC '97 Controller Theory of Operation

The ICH4 AC '97 Digital Controller (DC) interface is an implementation of the AC-link, with additional features to support the transaction and device power management. The ICH4 AC'97 DC includes DMA engines for high-performance data transfer to memory via a hub interface.

ICH4 AC'97 DC and link supports isochronous traffic, which emphasizes the timing of the data. This is critical to maintain the data stream from the audio and/or modem codec.

2.1. Intel® ICH4 AC '97 Initialization

2.1.1. System Reset

The ICH4 AC '97 circuitry is reset on power up by combining the PCIRST# signal with the AC Link RESET# signal. However, AC Link RESET# will not follow PCIRST# during a resume from sleep condition. During operation, the system can be reset by clearing the AC '97 cold reset bit in the Global Control/Status register (GLOB_CNT). This bit is maintained during ICH4 sleep mode and can be used by the driver to select warm or cold reset during a resume condition. If the codec is not present, i.e. ICH4 AC '97 is not supported, codec ready will never be seen by the controller. Once the reset has occurred, a **read** to Mixer register 00h/80h will indicate what type of hardware resides in the codec(s).

Software must ensure that codec ready bit is present for the appropriate Global Control/Status register (GLOB_CNT). Before writing any value on the codec registers or initiating a DMA transfer, s/w must ensure that the analog portion of the codec has reached a ready status by reading the Audio codec register Powerdown Control/Status register (Index 26h) or Extended Modem Status and Control register (Index 3Eh) correspondingly.

2.1.2. Codec Topology

The following rules present the allowable codec configuration when attaching to the AC-Link interface. To avoid improper driver loading, the system BIOS should determine the presence of the audio or modem codec attached on the AC-link, and enable the Audio or Modem function's PCI configuration space accordingly.

The following are the loading rules for ICH4:

1. Maximum of three codecs total on the link
2. Maximum of a single modem function, either as Modem Codec or a combination Audio/Modem Codec

This information is used to disable (hide) the appropriate PCI function. To determine that a codec or codecs are attached to the link, the System BIOS follows an algorithmic approach

Drivers can distribute output and input data in appropriate slots on available codec. For example a 6-channel data stream can be separate into three 2 Channel codecs as long as the codecs are programed to decode the appropriate slot output stream (SDATA_OUT). Similarly ICH4 provides 2 Stereo PCM input channels as well as 2 Microphone

mono input DMA channels. These allow for separate input streams for stereo PCM and microphone recording from two different codecs simultaneously.

Software should match sample rates, when two codecs are teamed together. The codecs must have matching vendors, types, and be explicitly supported in software. Essentially, audio codecs must be programmed with a common sample rate. The selection of a common sample rate is based on each codec's capabilities. As detailed in section

2.1.3. BIOS PCI Configuration

The ICH4 AC '97 Controller as previously indicated, exposes two PCI functions in the ICH4 (Bus 0, Device 31h). This allows for driver differentiation between these capabilities in the component.

- Function 5: ICH4 AC '97 Audio Controller
- Function 6: ICH4 AC '97 Modem Controller

As PCI devices there are a number of registers that are required to be initialized to enable these functions. The following table summarizes these requirements.

Table 2-1. Audio Registers

Device 31 Function 5 Audio			
Offset	**Register**	**Default**	**Comments**
04h-05h	Command (COM)	0000h	Bit 2: Bus Master Enable Bit 1: Memory Space Enable Bit 0: When enable in 41h I/O Space Enable
10h-13h	Native Audio Mixer Base Address (NAMBAR)	00000001h	When enable in 41h Address in the 64K I/O space that allows 256 bytes of registers not in conflict with any other set
14h - 17h	Native Audio Bus Mastering Base Address (NABMBAR)	00000001h	When enable in 41h Address in the 64K I/O space that allows 256 bytes of registers not in conflict with any other set
18h – 1Bh	Memory Audio Mixer Base Address (MMBAR)	00000000h	Address in the 4-GB memory space that allows 512 Bytes of registers not in conflict with any other set
1Ch – 1Fh	Memory Bus Master Base Address Register (MBBAR)	00000000h	Address in the 4-GB memory space that allows 512 bytes of registers not in conflict with any other set
3Ch	Interrupt Line (INTLN)	00h	A hardware interrupt (0-Fh) that follows value assigned to PIRQB#. Has not effect on ICH4 it is used to indicate software the IRQ value assigned to the device.
41h	CFG Configuration	00h	When cleared, the I/O space BARs at offset 10h and 14h become read only registers. This is the default state for the I/O BARs. Initialize by BIOS when backward I/O Bar compatibility is required Memory BARs are always enable.

Table 2-2. Modem Registers

Device 31 Function 6 Modem			
Offset	**Register**	**Default**	**Comments**
04h-05h	Command (COM)	0000h	Bit 2: Bus Master Enable Bit 0: I/O Space Enable
10h-13h	Native Audio Mixer Base Address	00000001h	Address in the 64K I/O space that allows 256 bytes of registers not in conflict with any other set
14h - 17h	Native Audio Bus Mastering Base Address	00000001h	Address in the 64K I/O space that allows 256 bytes of registers not in conflict with any other set
3Ch	Interrupt Line (INTLN)	00h	A hardware interrupt (0-Fh) that follows value assigned to PIRQB#. Has not effect on ICH4 it is used to indicate software the IRQ value assigned to the device.

With the exception of register 41h on Device 31 Function 5, initialization of the PCI registers above is the responsibility of the PnP capable OS. If a PnP OS is not available in the system, it is then the BIOS's responsibility to configure all PCI devices including the registers above. Determination of the presence of PnP capable OS is usually made via a switch in the System Setup. However, the final configuration or the existence or not of this switch is implementation dependent.

The ICH4 AC'97 controllers also provide PCI Power Management functionality. PCI Power Management Registers are available via the configuration space. Handling of the Power Management registers is responsibility of the OS PCI Bus driver following standard procedures. For further discussion on the usage model for this registers please review the power management section of this document.

2.1.4. Hardware Interrupt Routing

The audio and modem functions in the ICH4 internally share the same PCI IRQ (PIRQB#). The configuration software must take this into account and assign the same IRQ pin to both functions. Sharing IRQs increases the ISR latencies. Each ISR must determine if the interrupting device is the one serviced by the routine, as determined by the OS programming model. PIRQB# it is also exposed as a PCI IRQ.

In an environment where a high Quality of Service (QoS) is required, system designers must pay close attention to devices attached to the same PIRQ. Software driven signal processing functions, as in the case of software driven modem and audio, require maintaining a low latency interrupt service in order to maintain proper functionality. Software driver programmers need to pay close attention to the ISR latencies and make use of Deferred Procedure Calls (DPC) as much as possible.

2.2. DMA Engines

The ICH4 AC '97 Controller uses a scatter gather mechanism to access memory. There are five, 16-bit DMA engines for Audio: 2 PCM Stereo In, 2 MIC mono in, and S/P DIF Out. There is one, 20-bit PCM 2/4/6 channel surround. There are two, 16-bit DMA engines for Modem: In and Out. Audio and Modem registers are located in two separate PCI functions in the ICH4 components to allow for driver development flexibility.

2.2.1. Buffer Descriptor List

The Buffer Descriptor list is an array of up to 32 entries, each of which describes a data buffer. Each entry contains a pointer to a data buffer, control bits and the length of the buffer being pointed to - the length is expressed as

number of samples. The buffer length is restricted to 65536 samples at 16 or 20 –bit per sample. A value of "0" in the buffer length indicates **no samples** to process. Each descriptor can point to a buffer of a different size.

Figure 2-1. Generic Form of Buffer Descriptor (One Entry in the List)

(Dword 0 : 00 -03h)

31 … 1	0
Buffer Pointer	0

(Dword 1 : 04 -07h)

31	30	29 … 16	15 … 0
IOC	BUP	R	Buffer Length

Table 2-3. BD Buffer Pointer (DWORD 0: 00-03h)

Bit	Description
31:1	**Buffer pointer**. This field points to the location of the data buffer. Since the samples can be as wide as 1 word, the buffer needs to be aligned to word boundaries to avoid having samples straddle dword boundaries.
0	**Reserved**. Must be 0 when writing this field.

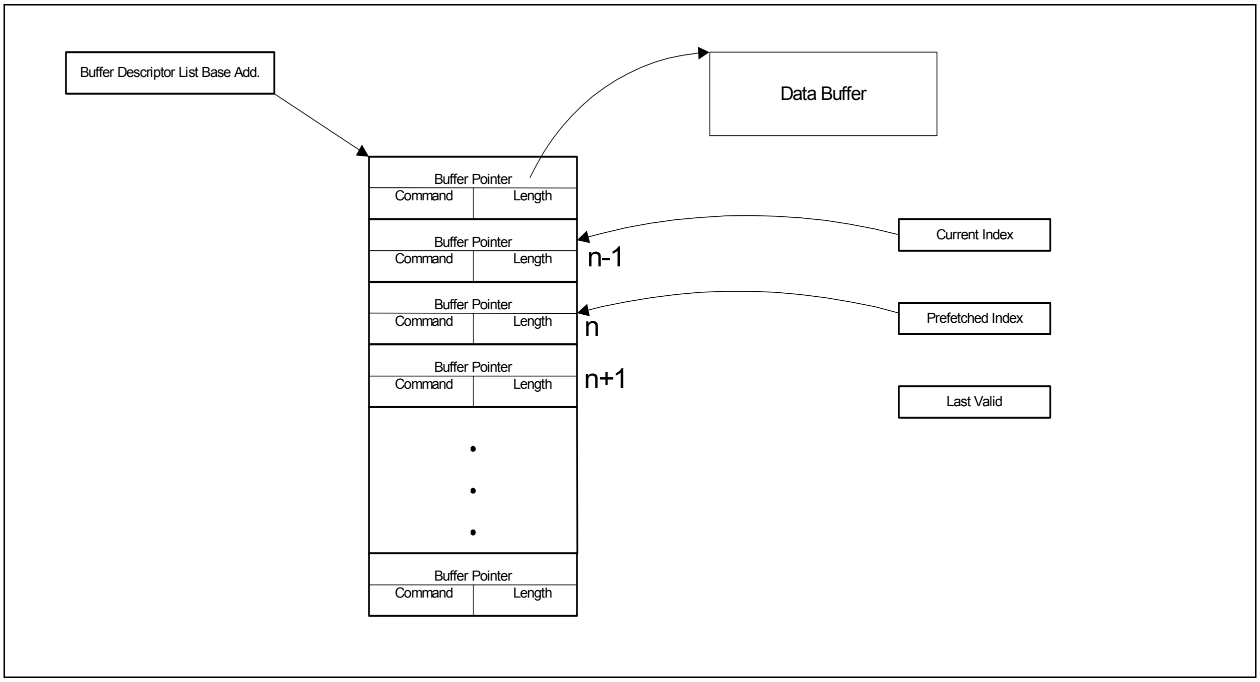
Table 2-4. BD Control and Length (DWORD 1: 04-07h)

Bit	Description
31	**Interrupt on Completion (IOC)** 1= Enable , 0 = Disable. When this it is set, it means the controller should issue an interrupt upon completion of this buffer. It should also set the IOC bit in the status register.
30	**Buffer Underrun Policy (BUP)** 0 = When this buffer is complete, if the next buffer is not yet ready, (last valid buffer has been processed) then continue to transmit the last valid sample. 1 = When this buffer is complete, if this is the last valid buffer, transmit zeros after this buffer is completely processed. This bit will typically be set only if this is the last buffer in the current stream.
29:16	**Reserved**. Must be 0 when writing this field.
15:0	**Buffer length**. This is the length of the data buffer in number of samples. The controller uses this data to determine the length of the buffer in bytes. A value of 0 indicates no sample to process.

2.2.2. DMA Initialization

The maximum length of the buffer descriptor list is fixed at 32 (this is limited by the size of the index registers). The figure below describes the organization of the Buffer Descriptor List.

Figure 2-3. Buffer Descriptor List



The following steps describe the driver initialization process for a single DMA engine. The same process should be repeated for each DMA engine.

1. Create the buffer descriptor list structure in memory (non-paged-poll).
2. Write the Buffer Descriptor List Base Address register with the base address of the buffer descriptor list.

Table 2-5. Audio Descriptor List Base Address

Audio Buffer Descriptor List Base Address	I/O Address
PCM IN	MBBAR + 00h (PIBAR)
PCM OUT	MBBAR + 10h (POBAR)
MIC	MBBAR + 20h (MCBAR)
MIC 2	MBBAR + 40h (M2DBAR)
PCM2 IN	MBBAR + 50h (PI2BAR)
SPBAR	MBBAR + 60h (SPBAR)

Table 2-6. Modem Descriptor List Base Address

Modem Buffer Descriptor List Base Address	I/O Address:
Line IN	MBAR + 00h (MIBDBAR)
Line OUT	MBAR + 10h (MOBDBAR),

3. Set up the buffer descriptors and their corresponding buffers. Buffers are passed to the miniport driver as Memory Descriptor Lists (MDL). These MDL describe the physical page address of the virtual audio buffer. Multiple buffer descriptors may be required to represent a single virtual buffer passed to the miniport driver

4. Once buffer descriptors are set in memory, software writes the Last Valid Index (LVI) register.

Table 2-7. Audio Last Valid Index

Audio Last Valid Index (LVI)	) I/O Address
PCM IN	MBBAR + 05h (PILVI),
PCM OUT	MBBAR + 15h (POLVI),
MIC	MBBAR + 25h (MCLVI)
MIC 2	MBBAR + 45h (M2LVI)
PCM2 IN	MBBAR + 55h (PI2LVI)
SPBAR	MBBAR + 65h (SPLVI)

Table 2-8. Modem Last Valid Index

Modem Last Valid Index (LVI)	I/O Address:
Line IN	MBAR + 05h (MILVI)
Line OUT	MBAR + 15h (MOLVI),

5. After LVI registers are updated, software sets the run bit in the control register to execute the descriptor list.

2.2.3. DMA Steady State Operation

Software has two concurrent activities to perform while in normal operation: Preparing new buffers/buffer descriptors and marking processed buffer descriptors and buffers as free. Once the run bit is set in the bus master control register bit 0, the bus master fetches the buffer descriptor.

1. Bus master starts processing the current buffer. Once current buffer is processed, depending upon the bits set in the command field, the interrupt is asserted and the interrupt bit is set.

2. Bus master increments the current and prefetch indices. It then starts executing the current buffer and schedules the next buffer to be prefetched.

3. Buffer service routine maintains a variable which points to the head of the list of descriptors to be processed. The descriptor list service routine performs the following activities:

```
// Update head of descriptors to be processed
     While (head != current_index)
     {
            Mark head free ;
            // check for end of descriptor list
            If head == base_address + (31 * 8);
                    // last entry on the list, set head to top of the list
                    head = base_address;
            Else
                    // still inside the list, increment head to next entry
                    head++;
```

Caution: **This algorithm needs to be optimized to reduce the number of memory accesses during execution. The "while" statement could translate to several memory access if this code is not execute after each buffer descriptor update.**

Also, the routine that prepares buffers maintains a variable that points to the entry *after* the tail of the list. This value is always the next entry after the Last Valid Index register. It follows the following algorithm:

```
// Update tail of descriptor list ready for execution and audio buffers //
when available for processing
     While ((tail == free) && (buffers_available > 0))
     {
            Prepare buffer descriptor indexed by tail;
            buffers_available--;
            //assign tail to Last Valid Index
            LVI = Tail;
            // check for end of descriptor list
            If (tail == base_address + 31 * 8);
                    // last entry on the list, set tail to top of the list
                    tail = base_address;
            Else
                    // Advance tail to next value
                    tail++;
            }
```

2.2.4. Stopping Transfers

There are two ways that transfers could be stopped.

1. By simply turning off the Bus Master run/pause bit. This will halt the current DMA transfer immediately. Data in the output FIFOs will continue to be read out until they empty. The registers will retain their current values and AC-link corresponding slots will be invalidated. Setting the run/pause bit will resume DMA activity.
2. Software can stop creating new buffers and hence not update the last valid index register. The bus master will stop once the last valid buffer has been processed. All register information is maintained. During this condition the controller will transmit the last valid sample or zeros pending the status of the Buffer Underrun Policy (BUP) bit in the buffer descriptor entry. If the run/pause bit remains set, then any future update to the Last Valid Index register will cause the bus master operation to resume.

Note: **Software must ensure that the DMA controller halted bit is set before attempting to reset registers.**

2.2.5. FIFO Error Conditions

Two general conditions could result in the FIFO error bit 4 in the status register being set. Pending the status of bit 3 in the control register it will also cause an interrupt.

2.2.5.1. FIFO Underrun

FIFO underrun will occur when the ICH4 AC '97 Controller FIFO is drained:

1. As a result of system congestion. The DMA read transaction could still be pending as data has not returned from memory. In this case the controller will repeat last sample until new data is available in the FIFO.
2. As a result of DMA engine reaching the Last Valid Index, no further access to memory, therefore FIFO will drain. In this case the controller will transmit the last valid sample or zeros pending the status of the Buffer Underrun Policy (BUP) bit in the buffer descriptor entry. This condition is an error if software is not able to update the descriptor list before the DMA engine reaches the Last Valid Index. However, this condition could be as result of the completing processing the last buffer. It is up to the software driver to determine the final status of this condition. See also Stopping Transfers in paragraph above.

2.2.5.2. FIFO Overrun

FIFO overrun will occur when valid data is transmitted in proper AC-link slots and DMA FIFO remains full. Two conditions could result in the FIFO error bit 4 in the status register being set. Pending the status of bit 3 in the control register it will also cause an interrupt.

1. As a result of DMA engine not being able to update system memory with the content of the FIFO. This is a result of system congestion. In this case, all new samples received from the AC-link will be lost.
2. As a result of the DMA engine reaching the Last Valid Index, no further access to memory, therefore FIFO will not drain. This condition is an error if software is not able to update the descriptor list before the DMA engine reaches the Last Valid Index. However, this condition could be the natural result of the last buffer entry been processed. It is up to the software driver to determine the final status of this condition. See also Stopping Transfers in paragraph above.

2.3. Channel Arbitration

It is possible for up to eight ICH4 AC '97 DMA channels to be enabled at one time. A round-robin arbitration scheme is used to arbitrate between these. channels.

2.4. Data Buffers

2.4.1. Memory Organization of Data

Samples are packed in two samples for 2 Channel (stereo), four samples for 4 Channels surround and six samples for 6 Channels surround.

The actual PCM data is "left-aligned" within the container. The sample itself is justified most significant; all extra bits are at the least-significant portion of the container. All non-valid data bits must be set to 0." With 20 bit data, the "top" 20 bits (31-12) of the DWORD contain the data. The bottom 12 bits (11-0) are not valid data.

The data must be sample aligned. 16-bit data would be WORD aligned and but 20 bit data is DWORD aligned, hence a 20-bit sample cannot start at the upper word of a DWORD.

2.4.2. PCM Buffer Restrictions

Below are the memory buffer restrictions for ICH4 PCM that applies for 2, 4, and 6 channel audio mode:

1. Buffer Descriptors Must Contain Integer Multiples of Framed Samples and Are Frame Aligned:

   Example: Two channel buffers must have a multiple of two samples. Four channel buffers must have 4, 8, 12, ... samples and six buffers channels have 6, 12, 18, … samples. The controller does not support a frame (e.g. left and right samples for 2 channel) spanning multiple descriptors. Similarly, the controller does not support a buffer descriptor with a single sample (PCM out MONO is not supported). Also, odd length buffers are not allowed due to the sample alignment requirements.

2. Software Is Allowed to Create an Empty Frame (0 Samples) in a Buffer Descriptor with the Following Restriction:

   An empty buffer has to be part of a list of buffer descriptors and it cannot to be the first Buffer Descriptor or the last Buffer Descriptor of the list. A series of buffer descriptors with 0 samples are possible in the lists as long as they are not the first or the last. The last Buffer Descriptor in the list is determined by the last valid index (LVI) that is programmed.

2.4.3. FIFO Organization

The ICH4 AC ’97 Controller supports 16-bit samples on all channels–with exception of PCM Out, which also supports 20-bit samples.

Data will be written to the FIFO in sample pairs following the order of valid slots in a channel. For example, for audio PCM in, the controller will check the first valid slot and add it to the FIFO first entry as a word (16 bits). The next valid slot will be added as the second word entry in the FIFO to create the PCM stereo sample pair. This behavior works under the assumption that the first valid slot will be always the Left channel (slot 3) followed by Right channel in slot 4 in the same or subsequent frame. If the codec transmits data repeating the slot, it will cause the controller to misplace the sample in the FIFO. Codecs compatible with the ICH4 AC ’97 implementation should always maintain the indicated order, and never use the same slot twice to transmit samples to the controller. The figures below present some ICH4 compatible and incompatible implementations using as a reference a two channel implementation.

Figure 2-5. Compatible Implementation with Left and Right Sample Pair in Slot 3/4 Every Frame

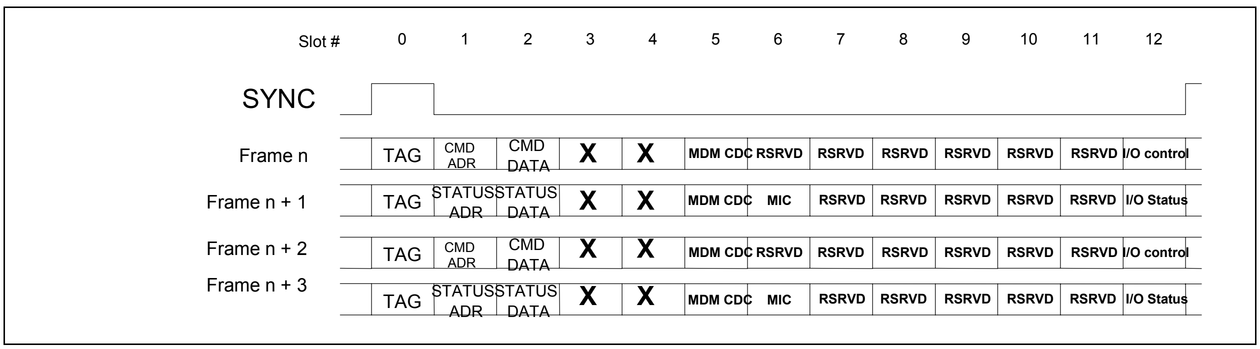


Figure 2-7. Compatible Implementation with Sample Rate Conversion Slots 3 and 4 Alternating over Next Frame

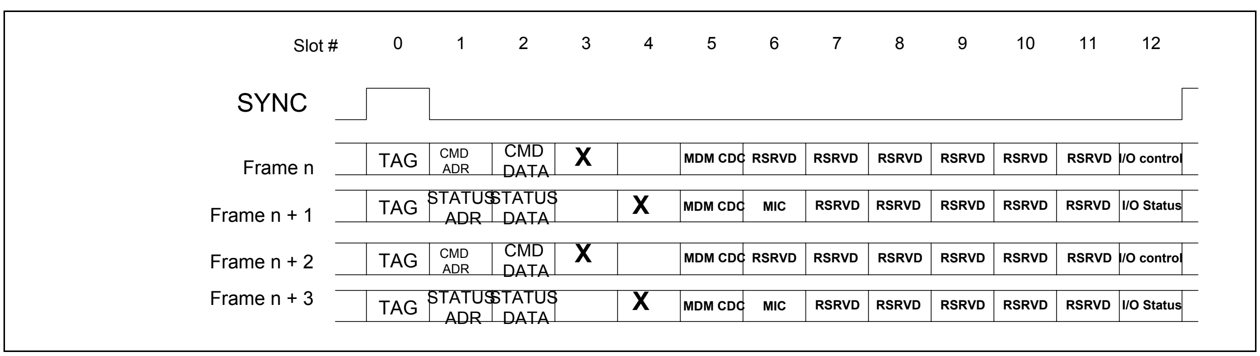


Figure 2-9. Incompatible Implementation of Sample Rate Conversion with Repeating Slots over Next Frames

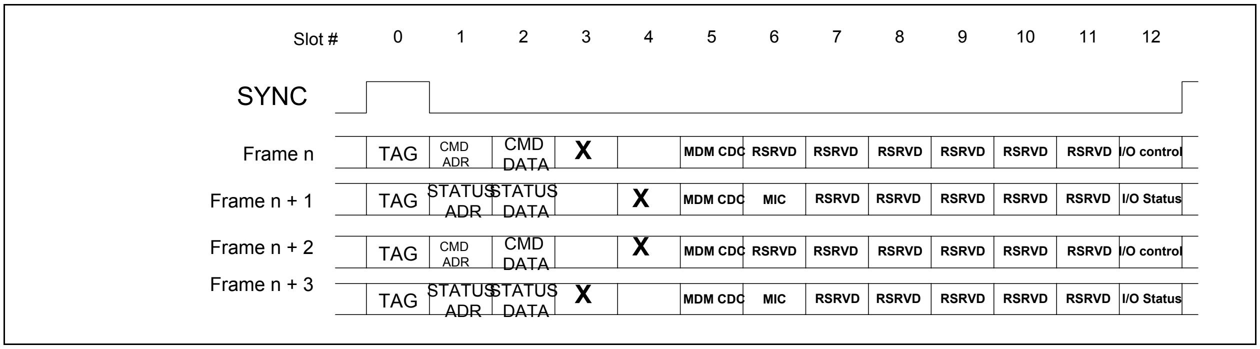


Table 2-9. FIFO Summary

Channel	# of Samples	FIFO Depth	FIFO Width	Comments
Mic In	1	2	32 bits	2 frame [1] dword
PCM In	2	4	32 bits	1 frames per dword
PCM Out	6, 4 or 2	24	32 bits	1 frames per dword (2 –16 bits ch), 1 frame per 2 dword (2 –20 bits ch) 1 frame per 2 dword (4 –16 bits ch) 1 frames per 4 dword (4 –20 bits ch) 1 frames per 3 dword (6 – 16 bits ch.) 1 frames per 6 dword (2 –20 bits ch)
Modem In	1	2	32 bits	2 frames dword
Modem Out	1	2	32 bits	2 frames dword

NOTES: 1. One audio frame worth of data for the specific DMA channel.

2.5. Multiple Codec/Driver Support

The ICH4 AC '97 Controller is capable of supporting a three-codec implementation. Under this implementation all codecs share the SDATA_OUT signal while independent SDATA_IN[0:2] are used by the codec to supply data to the controller. ICH4 allows for a compatible behavior, where the three SDATA_IN are used, these signals are logically OR'd inside the digital controller, effectively creating one digital input data stream. However, ICH4 also allows for an independent SDATA_IN functionality. On independent functionality, the SDATA_IN Map Register (SDM) MBBAR + 80h is used to steer the content of the input slots to the appropriate controller DMA engine. This capability also allows for a more reliable enumeration algorithm of the available codecs.

Table 2-10: SDM Register Description

Bit	Type	Reset	Description
7:6	RW	00	PCM In 2, Microphone In 2 Data In Line (D21L): When the SE bit is set, these bits indicates which SDATA_IN line should be used by the hardware for decoding the input slots for PCM In 2 and Microphone In 2. When the SE bit is cleared, the value of these bits are irrelevant, and PCM In 2 and Mic In 2 DMA engines are not available. 00 SDATA_IN0 01 SDATA_IN1 10 SDATA_IN2 11 Reserved
5:4	RW	00	PCM In 1, Microphone In 1 Data In Line (DI1L): When the SE bit is set, these bits indicates which SDATA_IN line should be used by the hardware for decoding the input slots for PCM In 1 and Microphone In 1. When the SE bit is cleared, the value of these bits are irrelevant, and the PCM In 1 and Mic In 1 engines use the OR'ed SDATA_IN lines. 00 SDATA_IN0 01 SDATA_IN1 10 SDATA_IN2 11 Reserved
3	RW	0	Steer Enable (SE): When set, the SDATA_IN lines are treated separately and not OR'd together before being sent to the DMA engines. When cleared, the SDATA_IN lines are OR'd together, and the "Microphone In 2" and "PCM In 2" DMA engines are not available.
2	RO	0	Reserved
1:0	RO	00	Last Codec Read Data Input (LDI): When a codec register is read, this indicates which SDATA_IN the read data returned on. Software can use this to determine how the codecs are mapped. The values are: 00 SDATA_IN0 01 SDATA_IN1 10 SDATA_IN2 11 Reserved

2.5.1. Codec Register Read

Codec register reads are presented in the AC-link in the next available frame after the controller receives the I/O transaction. Data will be returned to the controller pending codec availability. To avoid longer latencies than necessary, the codec must return data in the next available frame. Multiple frame transactions impose large system latencies, to the detriment of system performance.

Even when data is returned in the frame immediately after the read request is presented in the AC-link, the minimum latency is still on the order of 40 µs. To minimize the effect on the system caused by long latencies in the AC-link, software drivers **must** maintain a copy of the codec register in memory (shadow) and use this data instead of accessing the codec.

Shadowing in memory is effective as long as the codec does not change the value of the registers itself. Therefore, the status of the GPIOs configured as inputs on the most recent frame is accessible to software by reading the register at offset 54h in the modem codec I/O space. Only the 16 MSBs are used to return GPI status. Reads from 54h will not be transmitted across the link. Instead, data received in slot 12 is stored internally in the controller, and the data from the most recent slot 12 is returned on reads from offset 54h.

The Powerdown in codec offset 26h and 3Eh status registers are not supported by an automatic shadowing mechanism, as is the case for offset 54h. However, these registers are sparingly used. These registers are read only during power down status determination.

Finally, codec ready status is required during system initialization. It is automatically reflected in the Global Status Register at MBBAR + 30h (*MBAR + 40h)* bit 8 for the primary codec, bit 9 for the secondary and bit 28 for the third codec. These three bits need not be saved in memory.

2.5.2. Codec Access Synchronization

All codec register writes are posted transactions in the ICH4 AC '97 Controller. The ICH4 AC '97 Controller will indicate transaction completion to the host processor immediately following the request even when the transaction is actually pending for completion in the AC-link. This is done to improve system performance. However, it also imposes restrictions in the driver(s) operation. Also, register reads present synchronization issues.

Before a codec register access is initiated, the driver must check the status of the Codec access in Progress (CAIP) bit 0 in the Codec Access Register at *MBBAR + 34h (MBAR + 44h.)* If no write is in progress, this bit will be '0' and the act of reading the register sets this bit to '1'. This reserves the driver the right to perform the I/O read or write access. Once the write is complete, hardware automatically clears the bit. The driver must also clear this bit if it decides not to perform a codec IO write after having read this bit. If the bit is already set, it indicates that another driver is performing a codec I/O writes across the link and the driver should try again later.

2.5.3. Data Request Synchronization in Audio Split Configurations

To support more than 2 channels of audio output, the AC '97 Component Specification, Revision 2.1 allows for a configuration where up to three audio codecs work concurrently to provide surround capabilities (refer to AC '97 Component Specification, Revision 2.2) To maintain data on demand capabilities the Intel Controller, when configured for 4 or 6 audio channels, will wait for **all** the appropriate Slot Request bits to be set before sending data in the SDATA_OUT slots. This allows for a simple FIFO synchronization of the attached codecs.

If the codecs on the link are not compatible, or are not known to be compatible, with respect to sample rate conversion algorithms and FIFO depth requirements (for instance, all codecs being the same revision of the same model from the same vendor), Variable Rate support should not be used, and a fixed sample rate of 48MHz is recommended to maintain synchronization across the codecs in use.

2.5.4. Bios Detection Algorithms

2.5.4.1. "Fast Codec Start" Codec Detection Algorithm

Referring to Figure 2-12, the following steps describe a method to detect and determine the type of the AC '97 codec(s) on the interface.

1. Enable the audio and modem functions of the AC '97 digital controller.
2. De-assert the AC '97 reset signal (AC_RESET#). In an effort to reduce the overall boot time of the system, the BIOS developer may consider executing other sections of the BIOS boot code while waiting for the AC '97 codecs to complete their power-up sequence and set the "codec ready" bits.
3. Wait for all codecs on the AC '97 Interface to become ready by monitoring the state of the "codec ready" bit.
   - 3a. The BIOS needs to time out after **800 uS**, if the "codec ready" bit is not set in order to prevent a potential lock-up condition with BIOS continuously waiting for a codec that does not exist.
   - 3b. If the BIOS times out while waiting for a "codec ready" bit to be set on the primary codec, then BIOS should not check for a secondary codec or tertiary codec.
   - 3c. If the BIOS times out while waiting for a "codec ready" bit to be set on the secondary codec, then the BIOS can assume that there is no secondary codec in the system.
   - 3d. If the BIOS times out while waiting for a "codec ready" bit to be set on the tertiary codec, then BIOS can assume that a primary codec is present and possibly a secondary codec (if a "codec ready" is set for the secondary codec).
4. Identify the codec function(s) of the primary codec (codec address 00b).
   - 4a. Read registers 02h and 3Ch of the primary codec. By using the values returned from registers 02h and 3Ch determine the functionality of the primary codec on the AC '97 Interface.
5. Identify the codec function(s) of the secondary codec (codec address 01b).
   - 5a. Read registers 02h and 3Ch of the secondary codec. By using the values returned from registers 02h and 3Ch determine the functionality of the secondary codec on the AC '97 Interface.
6. Identify the codec function(s) of the tertiary codec (codec address 10b or 11b).
   - 6a. Read registers 02h and 3Ch of the tertiary codec. By using the values returned from registers 02h and 3Ch determine the functionality of the tertiary codec on the AC '97 Interface.
7. If a primary codec was not found on the AC '97 interface, then the AC '97 controller must be disabled to ensure that the operating system does not attempt to load a driver.
8. If a primary codec was found, then the BIOS AC '97 codec detection routines can continue.

2.5.4.1.1.Standard Codec Detection Algorithm

Referring to Figure 2-13, the following steps describe a method to detect and determine the type of the AC '97 R2.1 codec(s) on the interface.

1. Enable the audio and modem functions of the AC '97 digital controller.
2. De-assert the AC '97 reset signal (AC_RESET#). In an effort to reduce the overall boot time of the system, the BIOS developer may consider executing other sections of the BIOS boot code while waiting for the AC '97 codecs to complete their power-up sequence and set the "codec ready" bits.
3. Wait for all codecs on the AC '97 Interface to become ready by monitoring the state of the "codec ready" bit.
   - 3a. The BIOS needs to time out after **600 mS**, if the "codec ready" bit is not set in order to prevent a potential lock-up condition with BIOS continuously waiting for a codec that does not exist.
   - 3b. If the BIOS times out while waiting for a "codec ready" bit to be set on the primary codec, then BIOS should not check for a secondary codec or tertiary codec.

3c. If the BIOS times out while waiting for a "codec ready" bit to be set on the secondary codec, then the BIOS can assume that there is no secondary codec in the system.

3d. If the BIOS times out while waiting for a "codec ready" bit to be set on the tertiary codec, then BIOS can assume that a primary codec is present and possibly a secondary codec (if a "codec ready" is set for the secondary codec).

4. Identify the codec function(s) of the primary codec (codec address 00b).

4a. Read registers 02h and 3Ch of the primary codec. By using the values returned from registers 02h and 3Ch determine the functionality of the primary codec on the AC '97 Interface.

5. Identify the codec function(s) of the secondary codec (codec address 01b).

5a. Read registers 02h and 3Ch of the secondary codec. By using the values returned from registers 02h and 3Ch, and referring to Table 5, determine the functionality of the secondary codec on the AC '97 Interface.

6. Identify the codec function(s) of the tertiary codec (codec address 10b or 11b).

6a. Read registers 02h and 3Ch of the tertiary codec. By using the values returned from registers 02h and 3Ch, and referring to Table, determine the functionality of the tertiary codec on the AC '97 Interface.

7. If a primary codec was not found on the AC '97 interface, then the AC '97 controller must be disabled to ensure that the operating system does not attempt to load a driver.

8. If a primary codec was found, then the BIOS AC '97 codec detection routines can continue

		AC '97 Codec Registers	
		Register 02h	Register 3Ch
Codec Function	Audio Codec (AC)	Non-zero value	Zero Value
	Modem Codec (MC)	Zero value	Non-zero value
	Audio/Modem Codec (AMC)	Non-zero value	Non-zero value

Figure 2-11 - AC '97 Codec Function Mapping to Codec Register Value

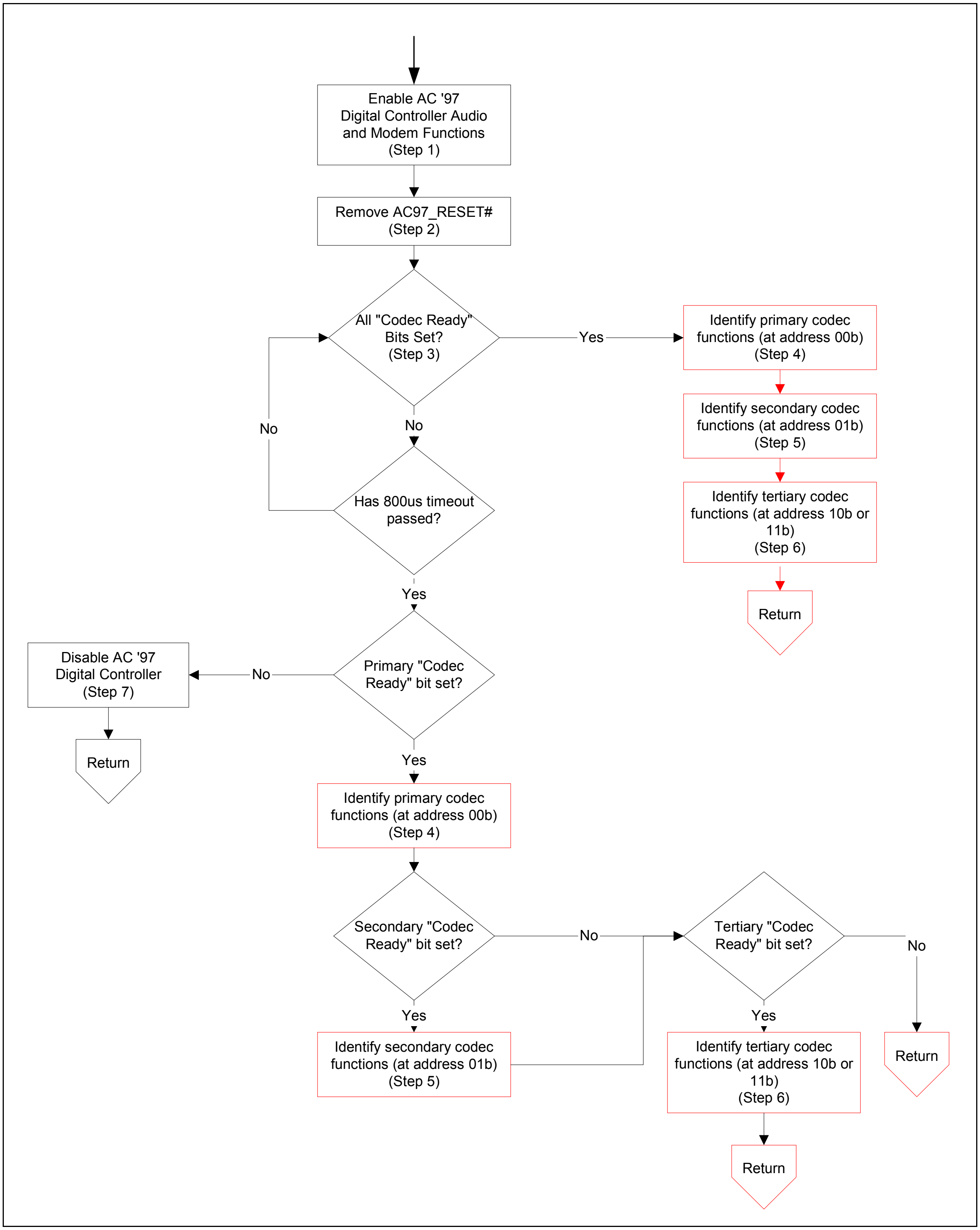


Figure 2-12 – AC '97 Presence and Function Detection Flowchart for Fast Codec Start Detection

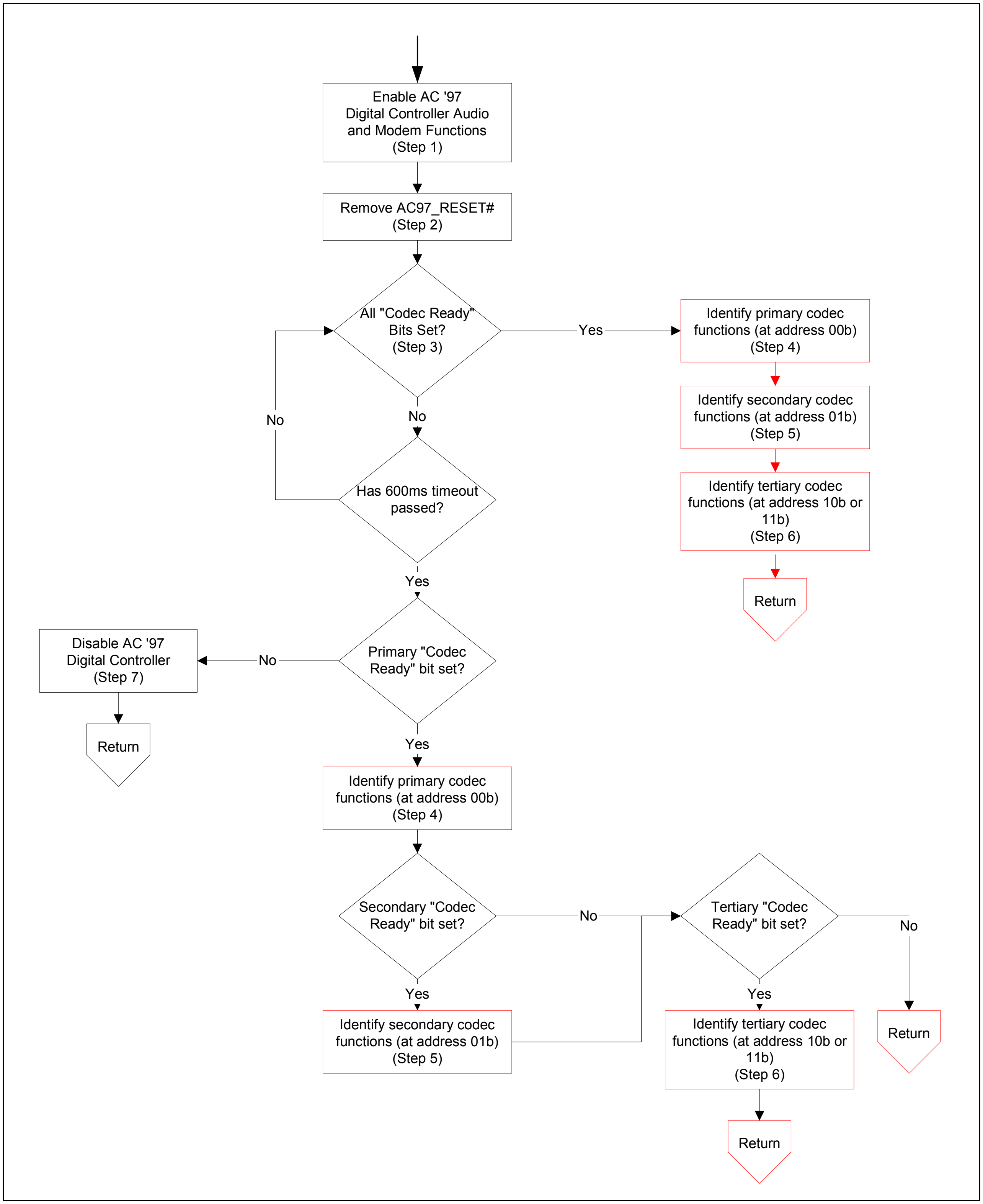


Figure 2-13 – AC '97 Codec Presence and Function Detection Flowchart for Standard Codec Detection

2.6. Power Management

Power management of the driver/codec interaction requires careful sequencing in the ICH4 AC ’97 environment. In the ICH4 AC ’97 environment it is possible to have two drivers sharing the same AC-link interface for two separate codecs or a single driver controlling two separate audio codecs. A driver forcing an aggressive sleep state in the link could have functional repercussions on the pairing codec. The Deep Sleep state in a device following ACPI compliance requirements is the D3 State. When a driver is requested to set its device to the D3 State, the driver should enter the most aggressive power saving mode possible. The D3 State is also often the precursor to a system wide core power removal. Therefore several considerations must be taken into account to maintain the device functionality and wake-up capability.

The procedure to be taken by an ICH4 AC ’97 device driver varies depending on the system configuration. The following table enumerates the possible codec combinations supported by the ICH4 AC ’97 Controller.

The ICH4 audio/modem controller supports a maximum of two codecs. The following system implementations are possible (see the figure above for details.)

Table 2-11. Codecs Topologies

Config.						
1	AC	(Primary)				
2	MC	(Primary)				
3	AMC	(Primary)				Possible D3 state interactions
4	AC	(Primary)	+	MC	(Secondary)	Possible D3 State interactions
5	AC	(Primary)	+	AC	(Secondary)	Driver interaction concern
6	AMC	(Primary)	+	AC	(Secondary)	Possible D3 State Interactions
7	AC	(Primary)	+	AMC	(Secondary)	Possible D3 State Interactions

Note: **The configuration above could be further limited by Intel ICH4 AC ’97 riser card configuration and loading. Refer to Audio/Modem I/O Riser Specification for details.**

It is evident that Configurations 1 and 2, above, require no driver synchronization between ICH4 AC ’97 codec’s. Configuration 1 and 2 are single codec topologies, and therefore an aggressive power saving mode is possible including disabling of the actual AC-link without concern of affecting paired codec functionality.

Configuration 3 is a single codec topology that provides both functions audio and modem. In this configuration driver interaction is also critical if a separate set of drivers are in control of the audio and modem functions.

Configuration 4, however, is a two-codec topology. In this configuration an aggressive power saving mode requires detailed attention to cross interactions and the effect on AC-link functionality.

Configuration 5 is a two-codec audio topology. In this configuration concerns are on the proper power down sequence. However, no driver interaction is expected as only the audio driver executes power management functions.

Configuration 6 is also a two-codec topology with split audio and integrated modem support. This is the most complex interaction, as two different sets of driver will be operating in a complex topology.

Configuration 7 is identical to configuration 6, with the primary and secondary codecs switched.

In order to power manage ICH4 AC '97 codecs, there are two sets of PR bits that drivers need to be managed. One set is at offset MMBAR + 26h in the audio function, mapped to offset 26h in the primary codec, and the second set is at MMBAR + 3Eh, mapped to offset 3Eh in the modem function. Notice that register 3Eh does not provide link down functionality, this is provided in register 56h bit 12, (MLNK) modem link.

2.6.1. Power Management Transition Maps

The following paragraphs relate power management transition maps in the constraints of an ACPI system environment. The tables below map codec PR bits transitions to specific ACPI "D" states for the device.

The following considerations were made in the generation of the following tables:

- Power management is defined in the framework of a desktop system. Further power savings are possible by implementing more aggressive power management typical of mobile environment policies (see aggressive power management section below). However these power savings are a trade off involving driver complexities and functional restrictions.
- Selection of a specific power policy below is pending the proper identification of the topology by the driver(s).
- Secondary codec is provided with external clocking mechanism and is not dependent on BIT_CLK to drive internal state machines when in power down mode.
- After warm or cold reset the device driver will bring all PR(x) bits to D0 state.
- Transition from/to any Dx state is accomplished by setting/resetting all appropriate PR(x) bit simultaneously. Codec should not place limitations on the PR(x) bits transition sequence represented above.
- Audio Codec Reg. 26h D15 EAPD (formerly PR<7> enable/disable function) is newly defined as control for an external audio power amp. Audio codec should provide an audio amp output pin (GPO) that provides off/on capability following this bit set/reset status.
- The modem tables assume Caller-ID capability during wake-up on ring, so Vref is on during D3.
- Modem D3 configuration is dependent on wake-up on ring event enable. If wake-up on ring is enabled, GPIO cannot go down in D3.

Note: When a codec section is powered back on the Powerdown Control/Status register (index 26h) should be read to verify that the section is ready before attempting any further operations.

Configuration Number 1: Single Audio Codec (Primary):

Table 2-12. Power State Mapping for Audio Single or Dual (Split) Codec Desktop Transition

PR<0:5> + (EAPD)								+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	EAPD	CLK	AC-Link	Mixer Vref.	Mixer	DAC	ADC					
Device State	7	5	4	3	2	1	0					
D0	0	0	0	0	0	0	0	On	On	On	On	All on
D1	0	0	0	0	0	***1***	***1***	On	On	On	On	-DAC, -ADC
D2	***1***	0	0	0	***1***	1	1	On	On	On	On	-Mix, -Amp

D3	1	***1***	***1***	***1***	1	1	1	Off	Off	Off	On	-Clock, -Vref

Configuration Number 2: Single Modem Codec (Primary):

Table 2-13. Power State Mapping for Modem Single Codec Desktop Transition

PR<A:D> + MLNK (other power control (PRx) bits do not apply for Intel® ICH4 implementation)						+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	Sdata_In	DAC1	ADC1	Vref	GPIO					
Device State	MLNK	D	C	B	A					
D0	0	0	0	0	0	On	On	On	On	All on
D1	0	***1***	***1***	0	0	On	On	On	On	-DAC, -ADC
D2	0	1	1	0	0	On	On	On	On	Same as D1
D3 (wake-up on ring)	***1***	1	1	0	0	Off	Off	Off	On	-Sdata_In,
D3	***1***	1	1	***1***	***1***	Off	Off	Off	On	-Sdata_In, -Vref, -GPIO

Configuration Numbers 3 to 6: Dual Function Single or Dual Codec Configurations:

Table 2-14. Power State Mapping for Audio in Dual Codec Desktop Transition

PR<0:5> + (EAPD)								+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	EAPD	CLK	AC-Link	Mixer Vref.	Mixer	DAC	ADC					
Device State	7	5	4	3	2	1	0					
D0	0	0	0	0	0	0	0	On	On	On	On	All on
D1	0	0	0	0	0	***1***	***1***	On	On	On	On	-DAC, -ADC
D2	***1***	0	0	0	***1***	1	1	On	On	On	On	-Mix, -Amp
D3	1	0	0	***1***	1	1	1	Off	Off	Off	On	-Clock, -Vref

NOTES:

1. PR(4) link down and PR(5) internal clocks disable are NOT recommended for desktop configuration. Setting these to power control bits could affect modem operation in an AC + MC configuration.
2. In a mobile system configuration, PR(4) and PR(5) could be used to provide further power savings. Driver designers should use D3 state codec semaphores in the ICH4 AC ’97 Controller to determine audio or modem codecs power status before setting PR(4) and PR(5) bits. Please refer to ICH4 AC ’97 External Architecture Specification for details. The miniport driver developed for the ICH4 AC ’97 Controller does not provide this capability.
3. In a dual audio codec transition, PR bits must be set in both of the codec registers. The primary audio power management register set is always located MMBAR + 26h, the secondary audio codec power management register set is located at MMBAR + A6h. Configuration software must sequence power down to the secondary codec first and then the primary codec. The process is reversed at resume when the primary codec is first restored and then the secondary.

Table 2-15. Power State Mapping for Modem in Dual Codec Desktop Transition

PR<A:D> + MLNK (other power control (PRx) bits do not apply for Intel® ICH4 implementation)						+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	Sdata_In	DAC1	ADC1	Vref	GPIO					
Device State	MLNK	D	C	B	A					
D0	0	0	0	0	0	On	On	On	On	All on
D1	0	***1***	***1***	0	0	On	On	On	On	-DAC, -ADC
D2	0	1	1	0	0	On	On	On	On	Same as D1
D3 (wake-up on ring)	***1***	1	1	0	0	Off	Off	Off	On	-Sdata_In,
D3	***1***	1	1	***1***	***1***	Off	Off	Off	On	-Sdata_In, -Vref, -GPIO

Table 2-14 and Table 2-15 above represent the recommended Power Transition Tables for a Desktop System. The tables above preclude the need for a driver to provide codec topology detection simplifying the initialization

sequence. These tables do not provide the maximum power saving. However, they are believed to provide sufficient power saving for the Desktop applications. The OEM and IHV are free to provide further differentiation by allowing the deeper power savings obtained by identifying the codec Topology.

2.6.2. Power Management Topology Considerations

A set of drivers could always assume Configuration Numbers 3 and 4, above, and establish their power management policy based on tables 2-15 and 2-14. These are the safest configurations with a semi-aggressive power management style consistent with a desktop environment. However, even in a desktop environment, further power savings are possible when in Single Codec Configurations Numbers 1 and 2. For the tables above to be implemented, the audio driver needs to be able to determine the AC-link topology configuration.

2.6.2.1. Determining the Presence of a Secondary and Tertiary Codec

To determine that a secondary codec is present, the driver needs to check both of the Codec Ready bits located in Global Status Register at:

Secondary Codec Ready   I/O Address: MBBAR + 30h (MBAR +40h)   bits 8,9 & 26

If two of these bits are set "1," it indicates that a secondary codec is active in the AC-link.

To determine that a Tertiary codec is present, the driver needs to check both of the Codec Ready bits located in Global Status Register at:

Tertiary Codec Ready :   I/O Address: MBBAR + 30h (MBAR +40h)   bits8,9 &26

If all of these bits are set "1," it indicates that a Tertiary codec is active in the AC-link.

2.6.2.2. Determining the Presence of a Modem Function

In the case of an AMC configuration, only the primary codec ready bit will be indicated. In order to determine proper power down configuration, the audio driver needs to determine the presence of modem functionality in the codec. The audio driver could check the Extended Modem ID register at:

Extended Modem ID: I/O Address: MMBAR + 3Ch

The content of this register will be FFh if no modem function is present.

2.6.3. Resume Context Recovery

When the system is placed in a power down state (S3 or greater) power is removed from the ICH4 AC '97 Controller. In this state DMA registers loose information regarding current position and pointer values. The device driver must be able to save the context of all DMA engines before acknowledging a D3 state. Device drivers must assume that D3 state request precedes a total system power lost therefore context for DMA engines. Upon system power resume, the device driver must restore the DMA engine register accordingly to saved values prior the suspend event.

2.6.4. Aggressive Power Management

As indicated in previous sections it is possible to go into a more aggressive power savings mode by carefully synchronizing audio and modem driver interactions over the AC-link. This aggressive power savings is usually found in mobile environments where battery power is critical.

Driver synchronization is required in a dual codec configuration where the audio driver could cause a link down power condition by setting the PR4 and PR5 bits in the audio codec register. When PR4 and PR5 are set, the AC-link base clock BIT_CLK is stopped. If this action occurs while the modem codec is still in operating mode, it will cause malfunctions and possibly a system hang.

To avoid this and similar situations, the audio and modem driver could follow a protocol using the provided audio and modem D3 state bit semaphores, AD3 for audio and MD3 for modem. These bits are located at:

Codec Write Semaphore Registers:

NABMAR +30h Audio I/O Space and MBAR + 40h Modem I/O Space

bit 16 for Audio (AD3)

bit 17 for Modem (MD3)

ICH4 AC '97 drivers should set the appropriate bit after setting the codec in a D3 state. The audio codec could use this semaphore to determine if the modem codec is already in a D3 state and shut down the link by also asserting PR4 and PR5 in the power management register in the audio function/codec. The following sections review in detail the sequence of events for drivers/codec entering a D3 state and a resume to D0.

2.6.4.1. Primary Audio Requested to D3

The audio power management procedure attempting to get the audio codec to D3 state.

```
If MD3 == true      // (sleeping?)
     {
            Audio_Power_Manage_Reg = D3 + PR4 + PR5;
            // yes, sleep plus AC Link down
     }
Else
     {
            Audio_Power_Manage_Reg = D3;   // No, sleep keeping link up
     }
AD3 = true;                                    // set to "audio sleeping"
// setting the flag last avoids race condition during D0->D3 transit.
```

2.6.4.2. Secondary Modem rRequested to D3

The modem power management procedure will try to get the modem codec to D3 state.

```
Secondary_codec = D3 + MLNK  // yes, sleep plus SDATA_IN1 low
MD3 = true
// setting the flag last avoids race condition during D0->D3 transit.
// MLNK corresponds to register Reg. 56h bit 12 (D12)
```

2.6.4.3. Secondary Modem Requested to D0

The modem power management procedure will try to get the modem codec to D0 state.

```
MD3 = false              // set to "modem awake"
//Setting the flag first avoid race condition during D3->D0 transit
If Modem_ready == True
     {
            Modem_Power_Manage_Reg = D0  // Bring back to fully awake,
     }
If AD3 == true          // (audio sleeping?)
     {
            Link_reset()      // cause a warm or cold reset
            While (!Modem_ready) // wait for modem ready
                     {
                          read modem codec ready bit every 400 ms
                     }
            Modem_Power_Manage_Reg = D0  // Bring back to awake,
     }
```

2.6.4.4. Audio Primary Requested to D0

The audio power management procedure will attempt to get the audio codec to D0 state.

```
AD3 = false              // set to "audio awake"
//Setting the flag first avoid race condition during D3->D0 transit
     If Audio_ready == True
            {
                     Audio_Power_Manage_Reg = D0;
                     // Bring back to fully awake,
            }
     If MD3 == true;          // (modem sleeping?)
            {
                     Link_reset();      // cause a warm or cold reset
                     While (!Audio_ready); // wait for modem ready
                           {
                                 read audio codec ready bit every 100ms;
                           }
                     Audio_Power_Manage_Reg = D0;  // Bring back to awake,
            }
```

Appendix C provides a schematic representation of the wake-up circuit. This should be used as reference for ACPI and APM wake up code as it relates with the paragraph above.

2.6.4.5. Using a Cold or Warm Reset

In the pseudo code above there are several references to resetting the AC-link using "Link_reset()". Drivers need to differentiate if the system enters a suspend event where core power is removed from the system before deciding to execute a Cold or Warm reset. A device is in a "D3 hot" state after the device is set in the lowest power consumption mode and core power is maintained. A device is in a "D3 cold" state when the device is set in the lowest power consumption mode and core power is removed.

In the ICH4 AC '97 implementation when core power is removed the cold reset bit is reset "0" This bit is located at:

MBBAR + 2Ch and MBAR + 3Ch

bit 1 **ICH4 AC '97 Cold Reset#.**

A driver requested to resume to a D0 state from a D3 state must check the status of the ICH4 AC '97 cold reset bit. If this bit has a value of "0" the driver will set it to "1" to de-assert the AC_RESET# signal in the link, thus

completing a cold reset. If the Cold Reset bit is set to “1” then a warm rest is required if the AC-link is down by the procedures indicated under aggressive power management. To execute an ICH4 AC ’97 warm reset the driver must set to “1” the ICH4 AC ’97 warm reset bit located at:

MBBAR + 2Ch and MBAR + 3Ch

bit 2 **ICH4 AC ’97 Warm Reset#**

A pseudo code representation is as follows:

```
void Link_reset(void)
      {
      If Cold_Reset# == True         // AC_RESET# asserted, D3 when cold!
            {
            Cold_Reset# = False;  // De-assert AC_RESET# Wake-up!
            }
      Else
            {
            Warm_reset = True;    // D3 is Hot! Do warm reset
            }
      }
```

This page is intentionally left blank.

3. Surround Audio Support

The AC '97 Component Specification allows for up to 6 channel of audio supported in the AC-link. The audio device driver must determine the number of audio channels available in the codec(s) and properly enable the ICH4 AC '97 Controller to support those channels in the link.

3.1. Determine Codec's Audio Channels

Upon determination of the link topology, system software must proceed to resolve the total number of audio channels available in the codec(s). Appendix A of the AC '97 Component Specification, Revision 2.1 defines codec register index 28h to indicate the presence of surround, center and LFE channels as follows: *Extended Audio ID Register (Index 28h)*

Table 3-1. Audio Codec Extended Audio ID Register

Bit D6:	CDAC=1 indicates optional PCM Center DAC is supported
Bit D7:	SDAC=1 indicates optional PCM Surround DAC is supported
Bit D8:	LDAC=1 indicates optional PCM LFE DAC is supported
Bit D9:	AMAP=1 indicates optional slot/DAC mappings based on Codec ID (Refer to AC '97 Component Specification, Revision 2.1, Appendix D for description)
Bit D15-D14:	ID1, ID0 is a 2-bit field which indicates the Codec configuration: Primary is 00; Secondary is 01, 10, or 11 (Refer to AC '97 Component Specification, Revision 2.1 Appendix C for details)

The AC '97 Component Specification, Revision 2.2, Section 5.8.2, provides detailed information on the usage model for multiple audio channels. This specification assumes the proper use of the AMAP bit describes the partition of the channels exposing critical functionality. If AMAP bit is not set to "1" a generic driver will not be able to determine the proper codec to slot distribution for a split audio codec configuration.

Based on Table 29 of the AC '97 Component Specification, Revision 2.2, the following table describes the available codec topology to audio channel distribution for ICH4 AC '97 Controller.

Table 3-2. Single Codec Audio Channel Distribution

Single Audio Codec Configuration	
Primary	**Total Audio Channels**
L/R	2
L/R; S-L/R	4
L/R; S-L/R; C/LFE	6

Table 33. Multiple Codec Audio Channel Distribution

Split Audio Codec Configuration			
Primary	**Secondary**	**Tertiary**	**Total Channels**
L/R	-	-	2
L/R	S-L/R	-	4
L/R	S-L/R; C/LFE	-	6
L/R	S-L/R	C/LFE	6

Legend:

L/R: Left Stereo Channel (Slot 3); Right Stereo Channel (Slot 4)

S-L/R: Surround Left Channel (Slot 7); Surround Right Channel (Slot 8)

C/LFE: Center Channel (Slot 6); Low Frequency Enhancement “subwoofer” (Slot 9)

(-): Not Applicable for this configuration could be used for simultaneous S/PIDF output

3.2. Enabling Intel® ICH4 AC '97 Controller Audio Channels

The ICH4 indicates how many channels supports in its Global Status Register. The ICH4 AC '97 Controller defaults to PCM Stereo after system reset or suspend-resume from S3, S4 or S5. Audio drivers can determine the total number of channels and re-configure the controller to support either 4 or 6 channel audio for PCM Out only. PCM In is not configurable and always is set for 2 -channels input. PCM MIC In is also not configurable and is always single channel (monaural.) The total numbers of audio channels supported are indicated at: *MBBAR + 30h: Global Status Registers.*

Table 3-3. CM 4/6 –PCM Channels Capability Bits

Bit	Description
21:20	**PCM 4/6 Capability:**: These read-only bits indicate the capability to support more than 2 channels for the PCM OUT. These bits will both be 1 to indicate that the Intel® ICH4 supports both 4 and 6-channel PCM output.

The ICH4 AC '97 Controller also provides support for an independent DMA channel to provide simultaneous S/PDIF output stream. This allows for concurrent PCM surround out with AC3 compress over S/PDIF interface. ICH4 AC '97 Controller follows slot assignments described in Section 5.4.2.1 and Table 13 of the AC '97 Component Specification, Revision 2.2.

System software enables the number of channels it intends to provide to the codec at:
MBBAR + 2Ch: Global Control Registers.

Table 3-4. AC-Link PCM 4/6 -Channels Enable Bits

21:20	PCM 4/6 Enable: Enables the PCM Output to be in 4 channel or 6-channel mode.	PCM Slots Enable: (PCM OUT DMA)	S/PDIF Slots Enable: (S/PDIF Out DMA)
	00 = 2 channel mode (default).	3, 4 (L and R)	7, 8
	01 = 4 channel mode.	3, 4, 7, 8 (L, R, RL and RR)	6, 9
	10 = 6 channel mode	3, 4, 7, 8, 6, 9 (L, R, RL, RR, C and LFE)	10, 11
	11 = Reserved	(undefined)	(undefined)

This page is intentionally left blank.

4. 20-Bits PCM Support

The ICH4 AC '97 Controller provides support for 16- or 20-bit PCM out. Software can determine if 20-bit samples are supported in the controller by reading the sample capabilities bits in GLOB_STA registers as follows: *MBBAR + 30h: Global Status Registers.*

Table 4-1. Sample Capabilities

Bit	Description
23:22	**Sample Capabilities:** Indicates the capability to support more greater than 16-bit audio. 00 = 16-bit Audio only supported (ICH1, ICH2, and ICH3 value) 01 = 16 and 20-bit Audio supported (ICH4 value) 10 = Reserved (for 24-bit audio support) 11 = Reserved

After determination of the controller capabilities, software can enable 20 –bits formats by programming the GLOB_CNT register as follows: *MBBAR + 2Ch: Global Control Registers.*

Table 4-2. PCM Out Mode Selector

Bit	Description
23:22	**PCM Out Mode (POM):** Enables the PCM out channel to use 16 or 20-bits. This does not affect the microphone, PCM In or S/PDIF DMA. When greater than 16-bits audio is used, the data structures are aligned as 32-bits per sample, with the upper order bits representing the data, and the lower order bits as don't care. 00 = 16 bit audio (default) 01 = 20 bit audio 10 = 24-bit audio (not supported. If set, indeterminate behavior will result. 11 = Reserved. If set, indeterminate behavior will result.

Note: Software must ensure that the PCM out DMA engine is stop and a new descriptor initialized before changing the value of PCM Out mode. These bits ***must not*** be modified when the PCM Out engine is running.

This page is intentionally left blank.

5. Independent S-P/DIF Output Capability

The ICH4 AC'97 controller provides an independent DMA engine for S-P/DIF output which operates independently of the six channel PCM Out stream. This allows the S-P/DIF data stream to be independent of the PCM stream, allowing usage models such as playing a DVD movie with output to S-P/DIF while at the same time using PCM In and Out in a telephony application.

The S-P/DIF DMA engine is initialized and is programmed in the same manner as the other DMA engines available, as described in section 2.2.

Data from the S-P/DIF DMA engine may be output on several different pairs of slots, depending on the codec configuration. The SSM bits in the Global Control Register control on which slots the S/P-DIF data is transmitted, as defined below. The default value (00) for this register is a reserved value, and so these bit MUST be appropriately programmed before the DMA engine is used. These bits are reset on controller reset, and are not affected by the reset of the S-P/DIF DMA engine.

Table 5-1, Global Control Register S-P/DIF Slot Map bits

Bit	Type	Reset	Description
31:30	RW	00	**S/PDIF Slot Map (SSM):** If the run/pause bus master bit (bit 0 of offset 2Bh) is set, then the value in these bits indicate which slots S/PDIF data is transmitted on. Software must ensure that the programming here does not conflict with the PCM channels being used. If there is a conflict, unpredictable behavior will result – the hardware will not check.

Bits	Output Slots for S/PDIF Data
00	***Reserved***
01	**7 & 8**
10	**6 & 9**
11	**10 & 11**

This page is intentionally left blank.

6. Support for Double Rate Audio

The ICH4 AC'97 controller has the capability of supporting a stereo 96KHz stream using the AC'97 Double Rate Audio (DRA) support. This capability is enabled by programming the controller to use four channel mode, which will place data on slots 3, 4, 7, and 8. A codec which is capable of treating the stream as PCM Left, Right, Left+1, and Right+1 data will interpret this as 96KHz stream, providing higher quality audio.

Note that the current AC'97 2.2 specification provide support provides the Double Rate date on slots 3, 4, 10, and 11, a mode which is not supported by the ICH4 controller. Codecs will need to additionally accept data on slots 3, 4, 7, and 8 to utilize DRA with the ICH4.

This page is intentionally left blank.

7. Independent Input Channels Capability

ICH4 AC '97 Controller provides capability for two new DMA channels dedicated to independent PCM and Microphone audio streams. These allow improved features that enable applications such as audio mobile docking and microphone arrays. Before the DMA transfer can occur software must determine the proper codec topology in the AC-link in order to program proper volume values and other relevant properties.

7.1. Link Topology Determination

While the SDATA_Out lines are shared by all AC '97 2.2 compliant codecs, the input DMA engines must be properly steered to specific SDATA_IN[0:2] lines on the link. Software must determine the specific codec ID assignment to the SDATA_IN[0:2] lines to properly adjust codec properties for operation. Software must fill a table similar to the following:

Table 7-1. Topology Descriptor

Input Channels	SDATA_IN[0:2]	Codec ID[00:10]
PCM IN & MIC	?	?
PCM IN 2 & MIC 2	?	?

Software uses a recurrent algorithm using the SDATA_IN Map Register to determine the values for the table above.

MBBAR + 80h: SDATA_IN Map Register.

Table 7-2. SDATA_IN Map

Bit	Description
7:6	PCM In 2, Microphone In 2 Data In Line (D21L): When the SE bit is set, these bits indicates which SDATA_IN line should be used by the hardware for decoding the input slots for PCM In 2 and Microphone In 2. When the SE bit is cleared, the value of these bits are irrelevant, and PCM In 2 and Mic In 2 DMA engines are not available. 00 SDATA_IN0 01 SDATA_IN1 10 SDATA_IN2 11 Reserved
5:4	PCM In 1, Microphone In 1 Data In Line (DI1L): When the SE bit is set, these bits indicates which SDATA_IN line should be used by the hardware for decoding the input slots for PCM In 1 and Microphone In 1. When the SE bit is cleared, the value of these bits are irrelevant, and the PCM In 1 and Mic In 1 engines use the OR'd SDATA_IN lines. 00 SDATA_IN0 01 SDATA_IN1 10 SDATA_IN2 11 Reserved
3	Steer Enable (SE): When set, the SDATA_IN lines are treated separately and not OR'd together before being sent to the DMA engines. When cleared, the SDATA_IN lines are OR'd together, and the "Microphone In 2" and "PCM In 2" DMA engines are not available.
2	Reserved
1:0	Last Codec Read Data Input (LDI): When a codec register is read, this indicates which SDATA_IN the read data returned on. Software can use this to determine how the codecs are mapped. The values are: 00 SDATA_IN0

Bit	Description
	01 SDATA_IN1 10 SDATA_IN2 11 Reserved

Software must follow the following steps to determine the codec topology:

1.- Determine codec present by reading the codec ready bits GLOB_STA Global Status Register.

MBBAR + 30h: SDATA_IN Map Register.

Table 7-3. Codec Ready Bits

Bit	Description
28	Tertiary Codec Ready (TRI) – RO. Provides the state of codec ready bit on SDATA_IN[2]
9	Secondary Codec Ready (TRI) – RO. Provides the state of codec ready bit on SDATA_IN[1]
8	Primary Codec Ready (PRI) – RO. Provides the state of codec ready bit on SDATA_IN[0]

2.- Once codec present is determined, software will determine the ID of the codec attached to each SDATA_IN line in used by reading any codec register at codec ID 00, 01 or 10. As indicated in Table 43 below and determine the returning SDATA_IN line by reading bits 1:0 in SDATA_IN Map registers as shown in Table 41 above.

Table 7-4. MMBAR: Mixer Base Address Register

Codec ID 00 (offset)	Codec ID 00 (Offset)	Codec ID 00 (Offset)
00h to 7Fh	80h to FFh	100-7Fh

3.- Base on the codec assignment and the specific application software will map the specific DMA engines to the appropriate SDATA_IN[0:2] lines by programming [7:6] for PCM/MIC and [5:4] for PCM/MIC2 in SDATA_IN MAP register as per table 23.

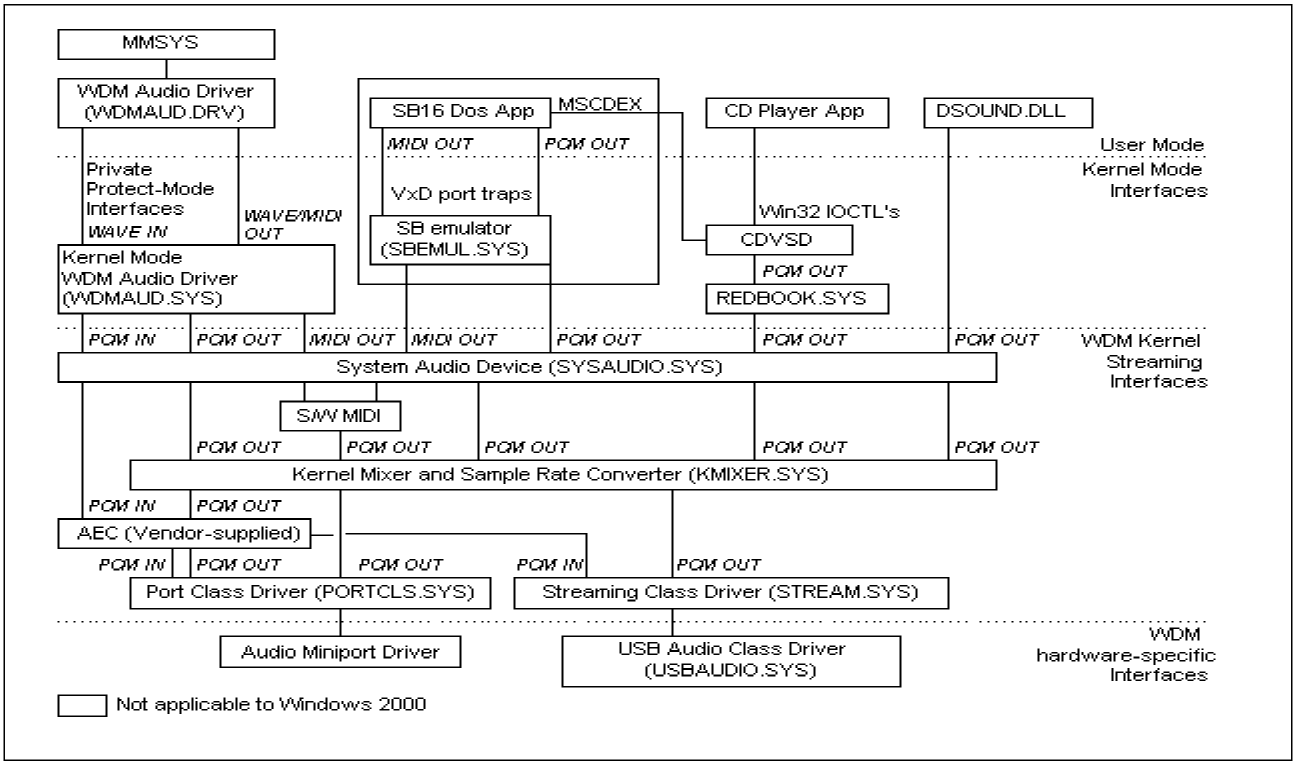
4.- Finally software will enable the steering mechanism by asserting bit [3] in SDATA_IN Map register before initiating a data transfer.

8. Intel® ICH4 AC '97 Audio Driver

8.1. Microsoft* Win32 Driver Model

The ICH4 AC '97 DC software interface is targeted to be implemented as a Microsoft* Win32 Driver Model (WDM) miniport driver. WDM allows for a common set of binaries for device classes and buses to be shared by the Microsoft* Windows* platforms that support this model, currently Windows* 98 Second Edition (SE) and Windows* 2000 operating systems.

Figure 8-1. WDM Audio Driver Hierarchy



The ICH4 AC '97 DC interface under WDM should be implemented as an audio miniport. Figure 8-1 illustrates how the different driver layers are organized and relate to each other. The port class driver is the highest driver in the chain that the miniport driver will communicate with. The PCI bus driver is responsible for enumerating the ICH4 AC '97 codec, providing the device's resources, and loading the miniport driver. The bus driver is supplied by Microsoft. The miniport driver, which is device dependent, is responsible for interpreting the commands received from the port class driver into the device specific functions. For details on the audio miniport please refer to:

Microsoft* Corporation, WDM Streaming Miniport Driver Model Specification Rev 0.1

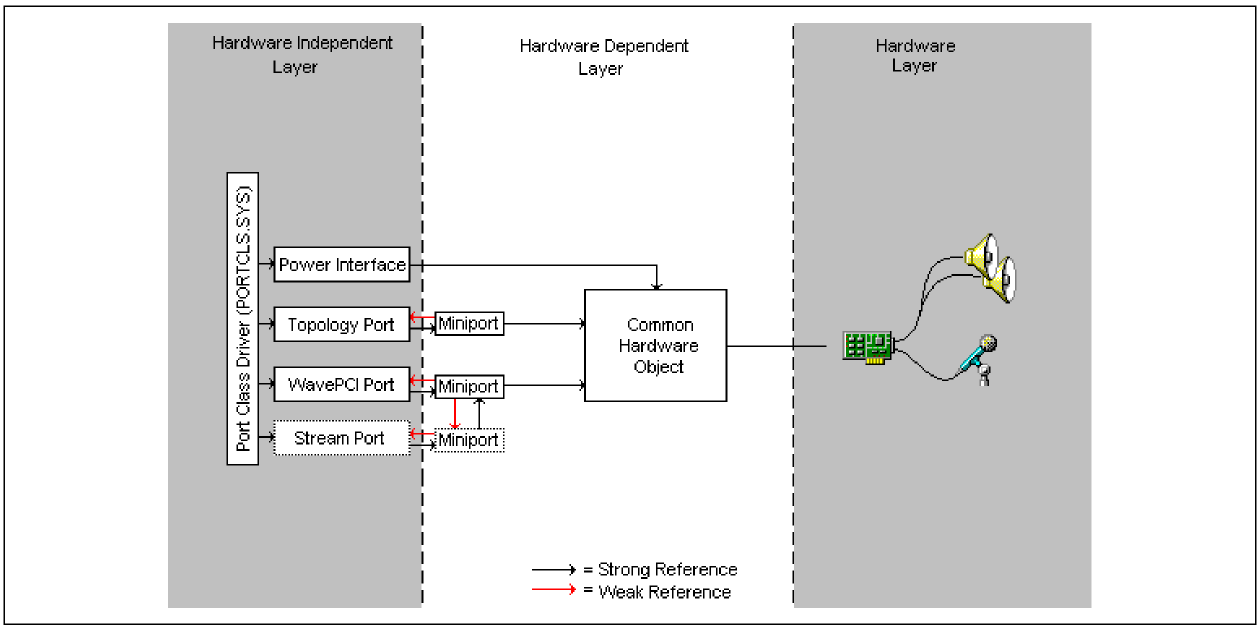
The ICH4 AC '97 driver supports methods for controlling the streaming clients of the port class driver and the analogue mixing capabilities of the AC '97 codec. In order to support the streaming clients, the driver creates a stream miniport, which allocates a buffer descriptor list for the scatter/gather DMA engine. The list is allocated from locked memory using HalAllocateCommonBuffer as the stream is initialized. Later, when the stream is started, the

stream miniport requests the initial packets from the operating system. The operating system provides both the physical and virtual addresses of the packets as well as the length of each packet in bytes. The driver the places the physical addresses and the lengths of the packets into the buffer descriptor list and starts the DMA. The driver also requests new packets during its interrupt service routine or when the operating system calls the stream miniport's MappingAvailable method to let it know that more packets are available. The analog mixing capabilities of the ICH4 AC '97 codec are exposed as a topology miniport. The operating system calls the miniport's property handlers in response to user requests.

8.2. Example Driver

The following description is provided as reference material to support an ICH4 AC '97 miniport driver. The driver was developed from the Creative Labs SoundBlaster* 16 example driver in the Microsoft* Windows* 2000 Device Driver Kit (DDK). As of Win2000 Beta3, the Microsoft* DDK http://microsoft.com/ddk/ also contains a sample ICH4 AC '97 miniport driver (src\wdm\audio\adapters\ac97).

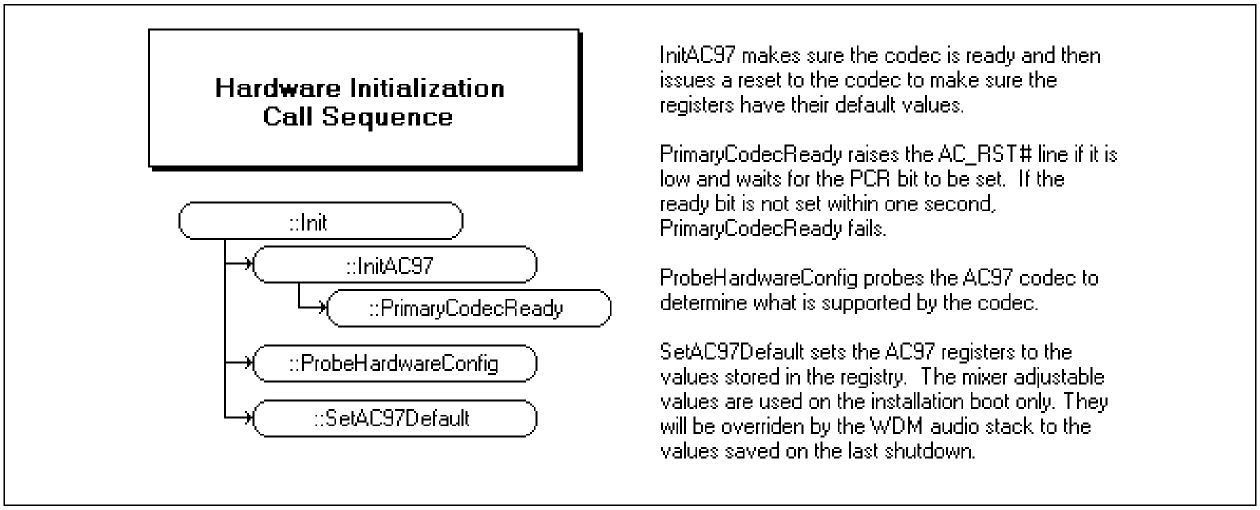
Figure 8-3. Driver Object Topology



The example driver contains four major C++ COM classes. Figure 8-3 (above) shows their relationship to one another and to the port class driver. This entire hierarchy is replicated for each enumerated device that references the driver. This is not to be confused with split codec configurations. The stream port/miniport pair is only instantiated when a new logical channel is created (e.g. every time we play a wave file).

ADAPTER.CPP contains the initial driver entry points. Once the resources (I/O ports and IRQ line) are claimed, the hardware object is instantiated. Further calls are made to the port driver to define the miniports' entry points.

Figure 8-5. Hardware Initialization Call Sequence



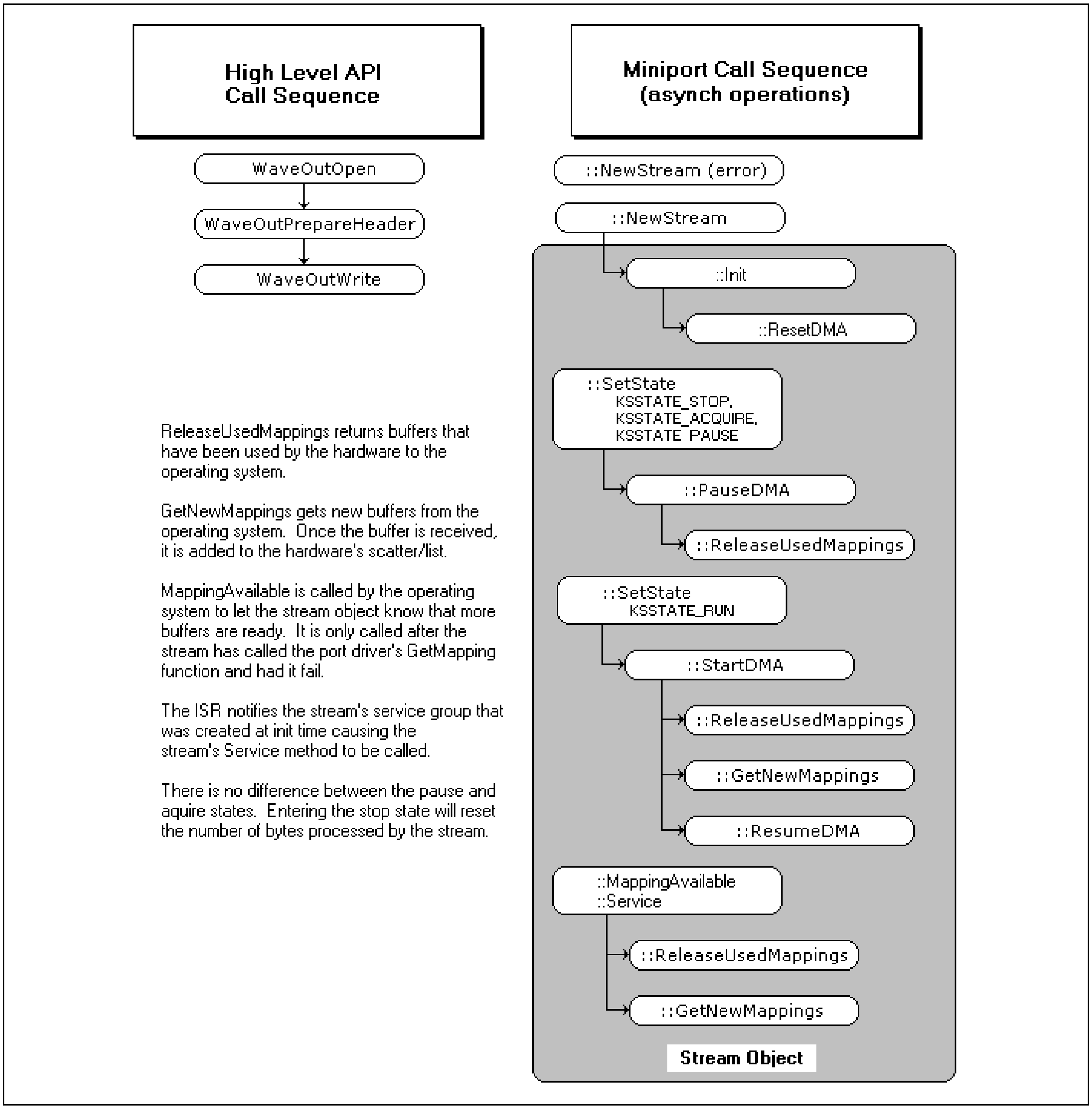
COMMON.CPP contains the definition of the common hardware object. This object is used to control all access to the physical hardware. Initialization of the physical hardware is accomplished by this object when it is initialized. The hardware initialization call sequence is outlined in Figure 8-5. The object also implements the IAdapterPowerManagement interface, which responds to ACPI messages from the operating system by saving and restoring the state of the codec.

MINTOPO.CPP contains the definition of the mixer topology object and implements the IMiniportTopologyICH interface (which includes the IMiniportTopology port driver interface). This object dynamically creates the mixer topology based on the results of probing the codec and returns the mixer property handlers’ entry points to the operating system.

PROPHND.CPP contains the code responsible for translating mixer property change requests into the ICH4 AC ’97 register values.

MINWAVE.CPP contains the definition of the WavePCI miniport object and implements the IMiniportWavePci and IPowerNotify port driver interfaces. This object is responsible for stream object creation, routing interrupts, and notifying the stream objects of changes to the device’s power state. When an interrupt occurs, the ISR determines the source of the interrupt and queue a deferred procedure call (DPC) to service the interrupt. The DPC will call the Service method of the stream object associated with the interrupt. A separate stream object is created for each DMA channel in the ICH. This object also determines the current position within the stream. *Care must be taken since the position is dependent on the current index and the current position in the buffer indicated by that index. These cannot be read in a single read and the current index may change before the buffer position is read.*

Figure 8-7. Stream Object Overview



ICHWAVE.CPP contains the definition of the stream object which implements the IMiniportWaveICHStream interface (which includes the IMiniportWavePciStream port driver interface). This object is controls the DMA of the ICH. This includes the responsibility of saving/restoring the state of the DMA in response to changes in the power state of the device. The interaction of the stream miniport with the operating system is outlined in Figure 8-7.

8.3. Plug and Play

The port driver filters all I/O request packets (IRPs) for the driver, including PnP IRPs. The device is started when the port driver is notified that the PCI bus driver has completed the IRP_MN_START_DEVICE IRP for the device.

Each time the device is started, the driver will create the driver object hierarchy. WDM audio drivers should minimize global data as this inhibits multi-device scenarios (this is not an issue with this hardware design). It is also good practice to allocate device instance data in response to starting the device as opposed to when the driver's AddDevice is called. This is because the driver must filter PnP IRPs (instead of letting the port driver filter them) if it wishes to be notified of when the device is stopped or removed.

When the device is stopped or removed, the port driver will release its pointers to the object interfaces. This results in a chain reaction, which will destroy the object hierarchy. As the destructors for the objects are called, all resources claimed by the device are released and any memory allocated by the driver is freed.

8.4. Power Management

Power management is supported under WDM through two optional port driver interfaces. The IAdapterPowerManagement interface may be used for general notification of changes in the devices power state. The OS may query the device prior to changing the power state of the device through this interface. This object that implements this interface is registered with the port driver when the device is started. The second interface is IPowerNotify. This interface may be implemented by miniport drivers if they wish to be notified independently of changes in the power state of the device. On transitions to lower power states, the port driver will call the methods of the IAdapterPowerManagement interface prior to calling the methods of the IPowerNotify Interface. The reverse is true for transitions to higher power states. In the ICH4 AC '97 example driver, both interfaces are implemented.

Since only one power management IRP is present in the system at any one time, the aggressive power management approach may be safely implemented and is encouraged. Multithreaded driver conflicts that would otherwise result are not an issue.

8.4.1. IAdapterPowerManagement

This interface is implemented by the common hardware object. Queries to change the power state of the device are never refused for any reason. On a transition to different power states, this driver will change the power state of the ICH4 AC '97 codec as outlined in the Power Management Transition Maps. There is no need to save the current state of the ICH4 AC '97 codec upon entering the D3 state as all codec writes have been cached by the driver to maximize performance. The registers of the ICH4 are not saved by the hardware object, as knowledge of the buffer descriptor list would be required. This responsibility is passed onto the WavePCI miniport object.

8.4.2. IPowerNotify

This interface is implemented by the WavePCI miniport object. The operating system is responsible for pausing any running streams prior to changing the device from the D0 state. On transitions to different power states, the miniport will use the weak reference to any stream objects to notify the stream of changes in the power state of the device. If no streams have been created, then the transition continues.

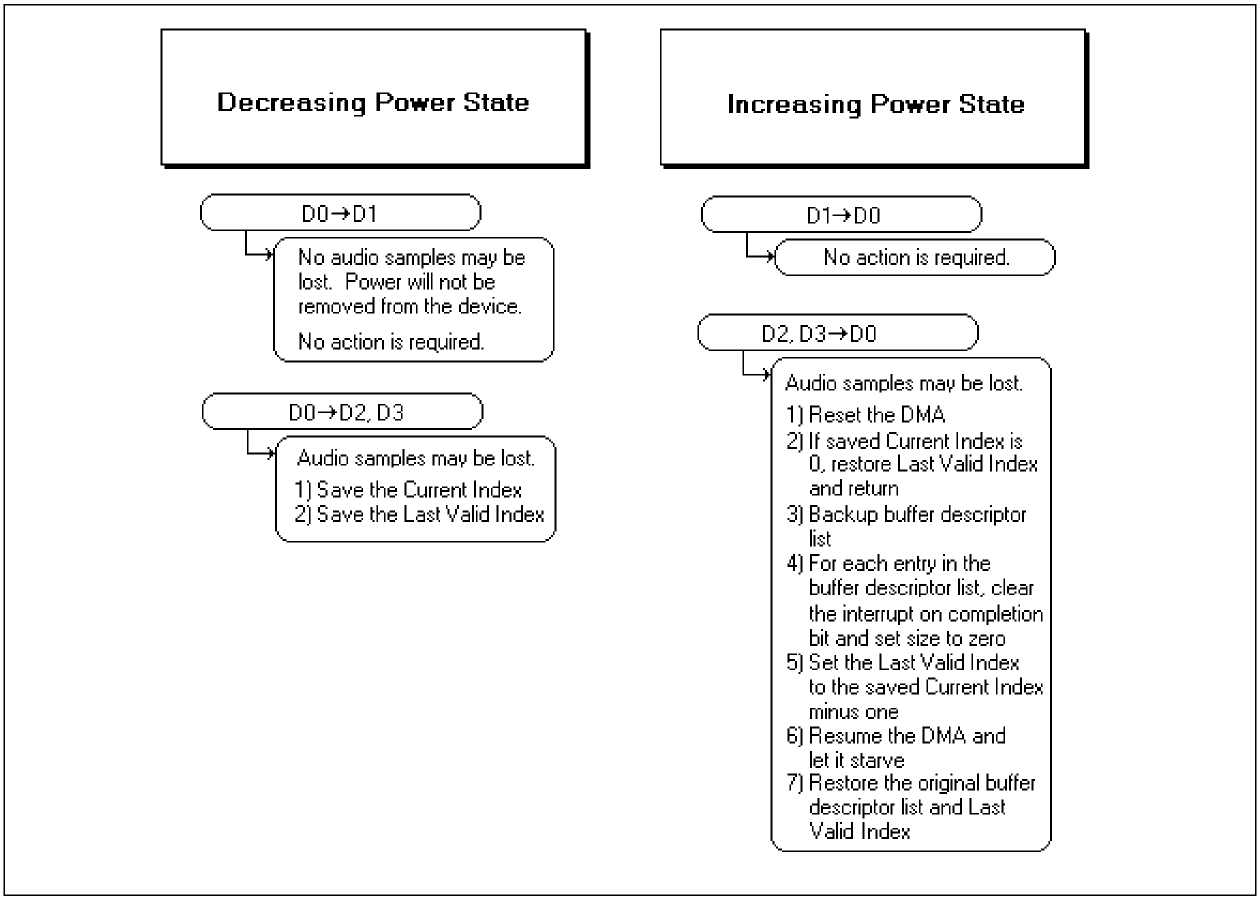


Figure 8-9. Intel® CH4 Power Transition Process

However, if streams are instantiated, the stream object must save/restore the state of its associated DMA channel. This is a complicated process as many of the ICH4 DMA registers are read-only. Figure 8-9 outlines the process followed by the driver for saving/restoring the ICH.

An alternative method would be to rearrange the buffer descriptor list if power is lost from the device. The chosen method attempts to restore the hardware to its previous state. The only register that is incorrect will be the position pointer within the current buffer. However, the amount is inconsequential and this limitation is not solved by rearranging the buffer descriptor list.

9. Intel® ICH4 AC '97 Modem Driver

The AC '97 Rev 2.2 specification allows for a modem codec to be connected to the AC-link interface. This allows for the development of a software stack that provides modem functionality, i.e. a soft modem. Currently there is no a single definition of how a soft-modem should be implemented under Microsoft* operating systems as is the case for audio in a WDM environment. Soft modem vendors have developed a variety of implementations for Windows* 95, Windows* 98, Windows NT* 4.0 and Windows NT* 5.0 operating systems. The design problems are not trivial for the soft-modem developer. This document does not attempt to describe solutions for each of these environments, but focuses instead on facilitating the development of the driver/hardware interface. At the time of this specification, Microsoft and Intel are engaging with the industry to define a common interface for the WDM environment.

9.1. Robust Host Based Generation of a Synchronous Data Stream

This section presents a method for synchronous modem data to be reliably generated on the host processor of a computer system where the host processor is running a non real-time operating system whose maximum response latency (interrupt, thread, etc.) exceeds the period at which the host processor generates consecutive buffers of modem data. For the purposes of this discussion we will assume that the host processor periodically, in response to interrupts, generates a buffer of modem data in memory which is then utilized or consumed in a synchronous fashion by hardware. This modem data would comprise a sequence of digital representations of the analog signal to be transmitted over a phone line in accordance with one of a variety of modem protocols, baud rates, etc., and could be transmitted to the ICH4 AC '97 DMA engines via the buffer descriptor list as described in Section 2.2.

We will also assume, for the sake of simplicity, that the data are double buffered, so that failure to generate new data before the next period will result in stream underflow from the hardware's viewpoint, but other scenarios including multiply buffered designs as well as non-periodic processing models can be accommodated. The algorithm works by providing good data followed by spurious data, which is chosen or computed so as to be adequate to maintain connection with the other modem by, for example, transitioning seamlessly with respect to the phase of the carrier frequency and the baud rate, thereby avoiding a retrain. This enables the datapumps of the two modems to maintain synchronization in the face of infrequent hold-offs from processing experienced by the datapump of the host-based transmitting modem. The spurious data will, of course, cause a packet retransmission or other action by the controller, but to the receiving modem the incoming data signal will be indistinguishable from one corrupted by line conditions.

The first invocation of the host based modem task provides an initial buffer and one or more buffers of spurious data (henceforth, spurious buffers). The task chooses or computes each of the spurious buffer(s) based on signal state at end of immediately preceding buffer. Note that these buffers do not have to be computed on the fly but can be precomputed and indexed into at run time. Subsequent invocations overwrite the previously provided spurious data with good data so that under normal conditions the spurious data is never used or consumed by the DMA engine. In the event that the host based modem task does not generate the next buffer in time for the DMA engine to begin consuming it the DMA engine is able to begin consuming the spurious buffer and thereby maintain seamless connection with the other modem's datapump.

9.1.1. Spurious Data Algorithm

The following pseudo code presents a conceptual view the algorithm. `LastState()` is a function of which returns a unique integer as a function of, for example, the carrier phase and baud position of the *last* sample in the buffer. In

an actual implementation this value would be computed during the course of generating the buffer. The `SpuriousBufferList` is an array of precomputed spurious buffers.

```
while (1)
{
    compute next buffer;
    pNextBuffer = &buffer;
    pSpuriousBuffer = &(SpuriousBufferList[LastState(buffer)]);
    wait for timer interrupt;
}
```

In this simplified scenario the device grabs the `pNextBuffer` address and stores it locally, using it to request the samples in the buffer one at a time. At the same time the device copies the `pSpuriousBuffer` into `pNextBuffer` so that when it is done with the current buffer it will get the spurious buffer unless the host software runs and overwrites `pNextBuffer` with a pointer to good data. In the next section we show how to implement the spurious data algorithm within the context of the ICH4 AC '97 buffer descriptor interface to hardware.

9.1.2. Intel® ICH4 AC '97 Spurious Data Implementation

The following pseudo code presents a modified version of the routine that prepares buffers and inserts them into the ICH4 AC '97 buffer descriptor list. In contrast to the version of this routine given in Section 2.2.3, in this version `tail` points to the last good (i.e., non-spurious) buffer in the list. Furthermore, because the ICH4 AC '97 DMA engine prefetches the next buffer descriptor we split the buffer generated by the datapump into two parts, with the second as small as practical and denote this size as `MinBufferLength` (here assumed to be 8 samples = 4 DWORDS = 500 µs. at 16 kHz.). For simplicity we assume that only a single buffer is generated by the datapump at a time and we ignore checking for the end of the descriptor list (i.e., the addition is implicitly modulo 32).

```
while (tail <= Prefetched_Index)
{
    tail++;  // Happens IFF Spurious Data was used
}
if (((tail <= LastValidIndex) || (tail == free)) &&
    (((tail+1) <= LastValidIndex) || ((tail+1) == free)))
{
    Descriptor.BufferPtr[tail] = &buffer;
    Descriptor.BufferLength[tail] = length(buffer) - MinBufferLength;
    Descriptor.BufferPtr[tail+1] =
        &buffer + length(buffer) - MinBufferLength;
    Descriptor.BufferLength[tail+1] = MinBufferLength;
    tail += 2;
 }
else
{
    ;    //error: no space for this data buffer
}
if ((tail <= LastValid index) || (tail == free))
    Descriptor.BufferPtr[tail] =
        &(SpuriousBufferList[LastState(buffer)]);
    Descriptor.BufferLength[tail] =
        SpuriousBufferLength[LastState(buffer)];
    LastValidIndex = tail;
    //Note: tail is NOT incremented, so next time we'll overwrite this
    //      descriptor, which is the whole point of this algorithm
}
else
{
    LastValidIndex = tail-1;//warning: no space for spurious data buffer
}
```

The above implementation can be improved upon in a number of ways. First, rather than adding a single (large) spurious buffer, a number of smaller ones could be chained together. In this way, the amount of spurious data actually transmitted would be reduced while still maintaining a given level of protection against long latencies for the host-based software. Additionally, the implementation could be extended to handle multiple buffers at once, inserting several buffers in a row and only splitting the last one and then appending a spurious buffer or buffers. Finally, the descriptor list is a circular buffer and a real implementation would have to check `tail` and `tail+1` against `base_address + 31 * 8`.